AFIPS

CONFERENCE PROCEEDINGS

1975

NATIONAL COMPUTER CONFERENCE

May 19-22, 1975
Anaheim, California

AFIPS PRESS
210 SUMMIT AVENUE
MONTVALE, NEW JERSEY 07645

The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1975 National Computer Conference or the American Federation of Information Processing Societies, Inc.

Library of Congress Catalog Card Number 55-44701
AFIPS PRESS
210 Summit Avenue
Montvale, New Jersey 07645

Printed in the United States of America

Composition of this volume was produced by Maryland Composition on a 32K Nova Computer (with two disks) which drive an 18 font VIP Photo Typesetter.

PART I

SCIENCE AND TECHNOLOGY

CONTENTS

MICROPROCESSORS

MICROPROCESSOR BASICS
Chairman: Rob Walker

MICROPROCESSORS AT WORK
Chairman: Paul M. Russo

BIPOLAR MICROPROCESSORS
Chairman: Theodore A. Laliotis

MICROPROGRAMMING AND MICROCOMPUTER PROGRAMMING

Chairman: Harut Barsamian

COMPUTER COMMUNICATIONS NETWORKS

COMPUTER COMMUNICATIONS—WHO, WHAT, WHEN, WHERE AND WHY?

Chairman: Howard Frank

ADVANCES IN PACKET-SWITCHING

Chairman: David C. Walden

ADVANCES IN PACKET RADIO COMMUNICATION

Chairman: Harry L. Van Trees

PACKET RADIO—FUTURE IMPACT

Chairman: Robert E. Kahn

SOFTWARE

PROGRAMMING AS AN ACT OF COMMUNICATION

Chairman: Robert Barton

Panelists: Anatol Holt, Charles Seitz, Robert Merrell, Lee Harrison III, William Huggins

SOFTWARE—PORTABILITY AND RELIABILITY

Chairman: David B. Wortman

PROGRAMMING—ART, SCIENCE OR ENGINEERING?

Chairman: Brian W. Kernighan

ISSUES IN PROGRAMMING LANGUAGE DESIGN

Chairman: Anthony I. Wasserman

COBOL '74—ITS IMPACT ON SOFTWARE ENGINEERING

Chairman: Paul Oliver

SOFTWARE ENGINEERING

Chairman: Thomas E. Bell

RELATIONAL DATA BASE IMPLEMENTATIONS

Chairman: David M. Dahm

RELATIONAL DATA BASE TECHNOLOGY

Chairman: Dennis W. Fife

QUERY LANGUAGES AND COMPARATIVE EVALUATION

Chairman: Ben Shneiderman

PERFORMANCE EVALUATION, DATA COMPRESSION AND SEARCH

Chairman: E. E. Lindstrom

DISTRIBUTED DATA BASES AND APPLICATIONS

Chairman: E. I. Lowenthal

STORAGE TECHNOLOGY

ENHANCING STORAGE RELIABILITY BY SOPHISTICATED CODING SCHEMES

Chairman: Jack Moshman

THE MASS STORAGE IMPACT

Chairman: John R. Morrison

ADVANCES IN NOVEL STORAGE TECHNOLOGIES

Chairman: John C. Davis

INTERACTION OF TECHNOLOGY AND SYSTEM ARCHITECTURE

INTERACTIVE GRAPHICS

FUTURE PROSPECTS IN DATA PROCESSING

FUTURE PROSPECTS IN DATA PROCESSING—II

Chairman: Orrin E. Taulbee

FUTURE PROSPECTS IN DATA PROCESSING—III

Chairman: Siegfried Treu

Panelists: Frederic G. Withington, David J. Farber, Orrin E. Taulbee, Thomas N. Pyke, Jr.

OTHER SCIENTIFIC AND TECHNICAL ASPECTS

WORKLOAD CHARACTERIZATION

Chairman: Kenneth W. Kolence

PSYCHOLOGICAL RESEARCH ON THE USE OF COMPUTER LANGUAGES

Chairman: James H. Carlisle

PART II

METHODS AND APPLICATIONS

USERS' VIEWPOINT ON EDP

MANAGEMENT AND COMPUTERS

DESIGN AND IMPLEMENTATION OF DISTRIBUTED SYSTEMS

Chairman: David J. Farber

MEDICAL AND HEALTH CARE COMPUTING

WHAT WENT WRONG WITH MEDICAL INFORMATION SYSTEMS—AN OPTIMISTIC OUTLOOK

Chairman: Jan F. Brandejs

COMPUTER APPLICATIONS IN AMBULATORY CARE

Chairman: Carlos Vallbona

MEDICAL INFORMATION SYSTEMS

Chairman: G. Octo Barnett

BANKING—TODAY'S CHALLENGES

INNOVATIVE APPLICATIONS OF COMPUTER SCIENCE

USER REQUIREMENTS

OTHER APPLICATIONS

COMPUTER AIDED MANUFACTURING

Chairman: Tony C. Woo

NEW APPLICATIONS IN PRINTING AND PUBLISHING

Chairman: Joseph J. Guiteras

COMPUTER SYSTEM SIMULATION AND PERFORMANCE EVALUATION

Chairman: Hans Kaspar

DATA BASES IN THE HUMANITIES

Chairman: James Joyce

ADVANCES IN COMPUTER TECHNOLOGY THROUGH AEROSPACE REQUIREMENTS

Chairman: Richard H. Thayer

PART III

INTERACTION WITH SOCIETY

EDUCATION—CURRICULA—TRAINING

MAKING COMPUTERS SAFER

LEGAL ASPECTS OF COMPUTER MANAGEMENT

OTHER SOCIETAL ASPECTS

GOVERNMENT FUNDING IN COMPUTER SCIENCE

Chairman: Kent K. Curtis

Advanced Research Projects Agency programs (Presentation only)

J. C. R. Licklider

National Institutes of Health programs (Presentation only)

William Baker

National Science Foundation programs (Presentation only)

Kent K. Curtis

Panelists: Richard L. Shuey, Keith Uncapher, Robert W. Ritchie

NATIONAL CENTERS FOR SCIENTIFIC COMPUTING

Chairman: G. Stuart Patterson, Jr.

The Institute for Advanced Computation (Presentation only)

Allan Birholtz

The National CTR Computer Center (Presentation only)

John Kileen

The National Center for Computation in Chemistry (Presentation only)

Peter Lykos

AFIPS PROGRAMS

Chairman: Paul W. Berthiaume

Guidelines to AFIPS support (Presentation only)

Paul W. Berthiaume

Programmers and system analysts job description projects (Presentation only)

Donn B. Parker

System review manual on security (Presentation only)

John Gosden

History of computing project (Presentation only)

William F. Luebbert

Washington activities study committee project (Presentation only)

Keith Uncapher

AFIPS privacy project (Presentation only)

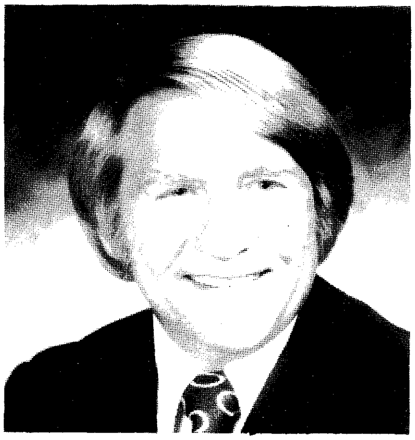
Willis H. Ware

INFORMATION PROCESSING—ITS IMPACT UPON SOCIETY THROUGH LIBRARY SYSTEMS

Chairman: Susan Crowe

COMPUTING APPLIED TO SOCIETAL PROBLEMS

Chairman: Donald L. Thomsen, Jr.



General Chairman:
Donal A. Meier
Consultant

Preface

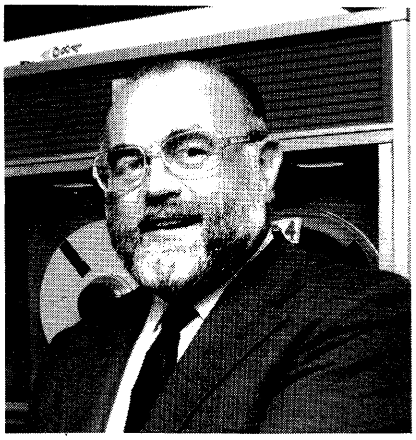
The purpose of the National Computer Conferences is to provide an atmosphere wherein designers, suppliers, users, managers, educators, and representatives of Government and Society at large can meet and interact. Discussions of new technical developments, as well as National and International issues and challenges facing the Information Processing Community, have been encouraged.

This year's discussions and developments are contained, for the most part, in this Volume 44 of the 1975 National Computer Conference.

The '75 NCC represents essentially all of the major computer-related Professional Societies. This year the Data Processing Management Association has become a sponsor. They join our other sponsors: The IEEE Computer Society, The Association for Computing Machinery, The Society for Computer Simulation and AFIPS. In addition, there are eleven other AFIPS Constituent Professional Societies which share an active role in helping to plan the National Computer Conference. The Institute of Internal Auditors has joined this group this year.

Along with the Technical Program, which is described in the Foreword, there are a number of other activities at NCC this year. They include four major invited addresses, a special "Pioneer Day Program," plus an Art Show, a Laser Show, a High School Computer Science Fair and Science Film Theater. There are approximately 250 companies and organizations participating in this year's Exhibition. These activities and exhibits, including all technical program sessions, will be held within the Anaheim Convention facility. Arrangements have also been made for a special "Day at Disneyland" during the Conference.

Volunteers, for a Conference of this magnitude, number in the hundreds. For the most part, they are members of the NCC Sponsoring Societies and the other AFIPS Constituent Societies. To these Societies and their participating members my heartiest thanks. To the NCC Board and the NCC Committee my thanks for your confidence and support. To all the members of the Technical Program Organization who contributed to the Sessions and made these Proceedings possible, thank you. And finally, to the AFIPS Headquarters Staff and all members of the '75 Conference Steering Committee, thank you for your dedication, time and effort. We did it!



Technical Program Chairman:
Stephen W. Miller
Stanford Research Institute
Menlo Park, California

Foreword

Those who shaped the program

The Technical Program Committee for 1975 was made up of active participants from the AFIPS sponsoring societies. This permitted representation from people with backgrounds in hardware, software, applications and social interaction. Each of the non-sponsoring constituent societies was requested, through their President and also their NCC Board Member, to appoint an active liaison to the Technical Program Committee. The sponsors and non-sponsoring constituent societies that assisted in the planning are listed in the back of this volume. All societies, sponsoring and non-sponsoring constituents, were then requested to make their session contributions within the National Computer Conference Program structure.

How the program is structured

The Program is structured into three major categories; Science and Technology, Methods and Applications, and Interaction with Society. Within each of these categories a number of technical areas were selected for special treatment. The Workshops and regional conferences from the Special Interest Groups and Technical Committees of sponsoring societies and conferences conducted by other constituent societies were used as an initial screening in the selection of topic areas which appeared of special importance to bring before a national forum with this diverse audience.

Each technical area was headed by a Director. These Directors were selected for their competence in the area, their demonstrated ability to organize a group of sessions articulating the major challenges and accomplishments of their area, and to provide broad representation of the major computer companies and universities. Most subject areas are developed as a "day" of four sessions in the same room. Many such days start with a tutorial session, continue with sessions of submitted papers, and terminate with a panel discussion led by leaders in the field. Hence, the order of presentation is important and that order within each area is preserved in the publication of this Proceedings.

Each paper selected, and published here, was judged on its individual merit by several of the referees whose names are listed at the end of the book. In some cases good papers were rejected because they did not fit this program. In a few cases a paper was accepted against the recommendations of the referees because it was a good statement of an important problem which formed the basis of the panel discussion. One paper included herein is nearly three times the nominal maximum length. We made an exception in this case since all reviews of this paper indicated that it was of exceptional quality and constituted "the definitive work" in its subject. The unanimous recommendation was that it should neither be shortened nor broken up into multiple papers.

While the program planners gave careful consideration to submitted papers, lack of volunteered papers did not restrict the planning. Several sessions consist entirely of invited papers, some sessions are rounded out by invited papers which complement submitted papers, and some sessions use one paper as a basis for developing a topic. When appropriate, the session may consist largely of visual presentations which do not lend themselves to publication.

Records of this conference: Proceedings, cassettes and monograms

This Proceedings constitutes the primary permanent record of the 1975 National Computer Conference. Information transfer at the Conference takes place not only through the papers recorded in this Proceedings, but also from panel sessions and in spontaneous interchanges following the presentation of papers. To capture a more complete record than is possible in the pre-published Proceedings, all sessions which have the consent of their participants are being recorded and tape cassettes will be made available for sale during and subsequent to the Conference. These cassette recordings supplement the published Proceedings in documenting the Conference.

A few select sessions will be transcribed, edited and published as separate AFIPS Monograms.

In another departure from precedent, the table of contents was expanded to include the names of all speakers and panelists and their topics, even though no text of their presentations appears in the Proceedings. In this way it serves as a pointer to the existence of the additional program material, as well as the table of contents of the Proceedings.

Of course, nothing substitutes for attendance at the Conference and active participation in the discussions. However, we believe that the combination of this Proceedings, the cassette recordings of the discussions, and the AFIPS Monograms constitute an excellent record of the Conference for those unable to attend, and an aid for the memory for those who do attend.

We hope that participation in this Conference has been both useful and enjoyable, and we express our thanks to the hundreds of participants who made it possible.

PART I

SCIENCE AND TECHNOLOGY

Area Director:
Theodore A. Laliotis
Fairchild Systems Technology
San Jose, California

Microprocessors

The term "microprocessor" as used today in the industry literature, has two possible meanings:

1. An LSI processor on a single chip.
2. A microprogrammable processor.

It is unfortunate from the literary standpoint that we use a word with two meanings; however, it is perfectly acceptable as a title for the series of sessions entitled "MICROPROCESSORS" at NCC 75 because both subjects are dealt with in these sessions.

LSI single chip processors are now becoming, very quickly, a mature design component from the standpoints of cost, availability, and performance. Microprogramming techniques have been around for a number of years as they were the prime innovation in third generation computers. The combination of the two makes a very powerful team which is undoubtedly the leading direction for the future of the industry. It is expected to cause the next major evolution, not just in the computer industry but in the total electronics industry much like the transistor did a couple of decades ago. The sessions are organized in a manner that presents both the hardware and software aspects of the state of the art.

The first session consists of three papers intended to address some of the fundamental topics of LSI microprocessors in a tutorial fashion utilizing state of the art components as examples. The first paper entitled "Microprocessor Rationale" examines some of the basic philosophies and design trade-offs and introduces the Signetics PIP microprocessor. The second paper presents the industry's first 16-bit LSI single chip microprocessor (National Semiconductor's PACE) from a user's viewpoint and explores applications that would benefit from such a product. The third paper addresses the important issues of I/O and data transfer techniques using Motorola's M6800 as an example.

The second session which is entitled "Microprocessors at Work" provides both an overview and specific examples of microprocessor applications in data communications, process control, numerical control, instrumentation and intelligent terminals.

The third session which is entitled "Bipolar Microprocessors" is intended to provide a forum for wider publicity and awareness of the state of the art developments in the area of bipolar LSI microprocessors. These devices are currently in the process of emerging and there is a large degree of interest in them due to their inherent high speed which is about one order of magnitude higher than the currently available MOS devices. Their prime applications will be in the implementation of high performance systems. Intel, Monolithic Memories, and others will present their cases in that session.

The fourth session is entitled "Microprogramming and Microcomputer Programming." This session treats the important considerations of the development and maintenance of software for microprocessor. Individual papers in this session include "An Emulation System for User Microprogramming," "Testing and Sequencing in Microprogrammed Computers," "Optimization Techniques for Horizontal Microprograms," and "Engineering and Maintaining Microcomputer Software."

The microprocessor rationale

by BRUCE THREEWITT
Signetics Corporation
Sunnyvale, California

THE MICROPROCESSOR RATIONALE

Classically, since the beginning of the computer era, logic system designs have been implemented with one of two entirely separate approaches. Economic factors determined whether hardwired logic or a computer would be used to perform the required operations. In many cases, the computer approach was far too costly.

Until recently, hardwired logic systems were constructed using large numbers of small-scale or medium-scale integrated circuits (SSI and MSI, respectively). Figure 1 includes examples of functions typically found in these two levels of integration. In general, the hardwired logic system could be modelled as a "black box" that takes inputs (variables/data) and performs some defined logic function on them, producing outputs (results) that are functions of the inputs and of time (see Figure 2). These outputs drive displays or other output devices that apply the results to doing work.

Meanwhile, the computer industry builds black boxes that accept inputs and perform some defined logic function to produce outputs that are functions of the inputs and of time (Figure 3a). The difference in these approaches lies in the contents of the respective black boxes. Computer systems have a certain minimum configuration as shown in Figure 3b. Until now, this configuration consisted of components and software that were too expensive to use in the simpler high volume logic system applications. As mini-computers tend downward in size and cost, more logic applications can be served; but, the cost/performance trade-offs of mini's still preclude their use in general hardwired logic replacement market.

As the need for less expensive, more versatile logic system design techniques continued to apply pressure to the digital electronic industry, a third system design alternative was evolving for the following reasons. First, logic designers were frequently going toward large-scale integrated circuits (LSI) to implement their particular system designs. Typically, a logic designer would take his logic and timing diagrams to an integrated circuit (IC) supplier and ask him to shrink that design onto one or a few MOS/LSI circuits (see Figure 4). He did this because he believed that the MOS/LSI technology offered a high-density, low power approach to digital circuits. Unfortunately, this assumption is not totally valid. It is true that the MOS/LSI technology yields small transistors. However, the MOS interconnection approach is significantly less dense than cells or devices. Random logic is characteristically dominated by complex interconnect with relatively few active devices (transistors). Thus, random logic designs cannot utilize the MOS technology efficiently. MOS/LSI does make very dense memories (patterned logic) where interconnects are minimized.

Secondly, the cost of producing custom MOS/LSI circuits is prohibitive in the long run both for the user and the IC manufacturer (Figure 5). This curve assumes that for random logic, increased complexity (integration level) results in increased specialization. When a logic system, which is specific to a given special purpose application, is integrated, the resulting IC's are also specific. Thus, the volume per product type decreases with increased integration. The few exceptions to this rule are single-application random logic devices such as calculator and watch circuits that generate sufficient volume to justify their existence. Also, the IC supplier must supply many different types of custom circuits in order to adequately serve the custom logic market, since any two given applications would probably not utilize the same custom circuits. This proliferation of circuit types greatly increases overhead costs for support engineering for testing and circuit design. Therefore, custom MOS/LSI, in general, is not economically feasible for the IC manufacturer or the systems manufacturer.

Thirdly, the appearance of the N-Channel silicon-gate MOS technology in recent times provides a vehicle for

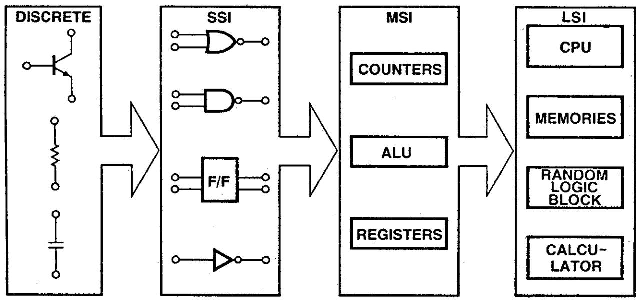


Figure 1—Trends in integration

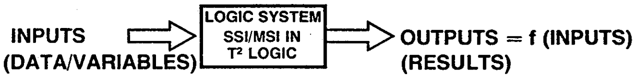


Figure 2—Logic system implementation I

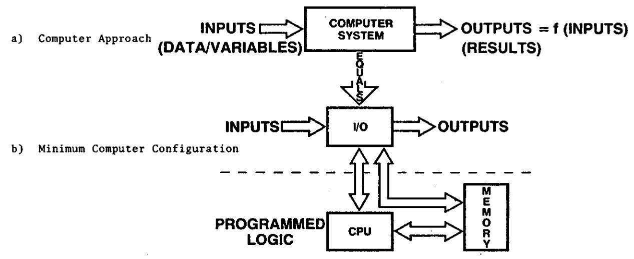


Figure 3—Logic system implementation II

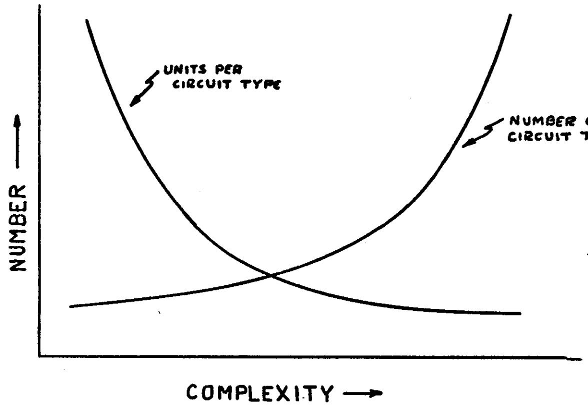


Figure 5—Custom random logic feasibility factors

building highly dense, low power, moderately fast, TTL-compatible LSI circuits. The resulting cost improvements are passed on to the system designer.

For these reasons, and since the IC supplier would like to exhaustively service the logic marketplace, a third alternative for random logic system designs has evolved. That alternative is a special case of random logic called programmed logic. Programmed logic has the characteristics outlined in Table I. Now, instead of wiring random logic together to perform a given function, the designer uses a general purpose logic block to perform logic according to the instructions contained in a program. Thus, the logic system designer will combine hardware and software techniques to achieve a system that was formerly all hardware. The resulting design is far more flexible since the features of the system can be a function of software (the program). When market conditions require an updated or even totally different system, in most cases only a new program need be written. The resulting improvement in system design turn-around time will significantly improve the flexibility of a system supplier in serving his market. If errors are made in the initial system design, corrections do not usually require a complete system redesign. Now, a custom system can be constructed with general purpose hardware by using a specialized program. The resulting savings in component costs alone, using readily available standard circuits, would justify this approach.

This programmed logic block can be constructed out of a new type of component called a *microprocessor*. More accurately, this component type is a micro-sized processor since it need not be micro-programmed. Perhaps a more appropriate name would be an Integrated Processor (IP). A general purpose processor is a logic device that can literally perform any computable function. From another point of view, a processor is a device that utilizes memory cells (used to store inputs and instructions) to perform logic functions.

Since breakthroughs in the various circuit technologies have generally resulted in increased density, the microprocessor allows the system designer to most efficiently utilize the state-of-the-art IC technology. Memories are widely and inexpensively available in many configurations, access modes, and cost-performance ranges.

On closer examination, the IP is an evolutionary extension of computer techniques for solving logic problems. Figure 3 shows a generalized block diagram of a minimum computer configuration. The IP is a miniaturized version of the Central Processing Unit (CPU) block. Thus, an IP is only part of a micro-sized computer. One must add memory and I/O (input/output) devices to construct a micro-sized computer.

Even though programmed logic uses computer techniques, the largest market for IP's is not in computer replacement. Rather, it is in hardwired logic replacement. Thus, in a few years, the huge Transistor-Transistor Logic (TTL) market will be largely serviced by IP's and memories instead. The IC industry is excited about integrated processors because, in replacing TTL, semi-conductor memory sales will increase because memories will be used in applications that have never before used them. Also, the IC supplier can now better service, in an economically feasible fashion, the market previously handled by custom LSI.

As integration levels increase, one is increasingly hard

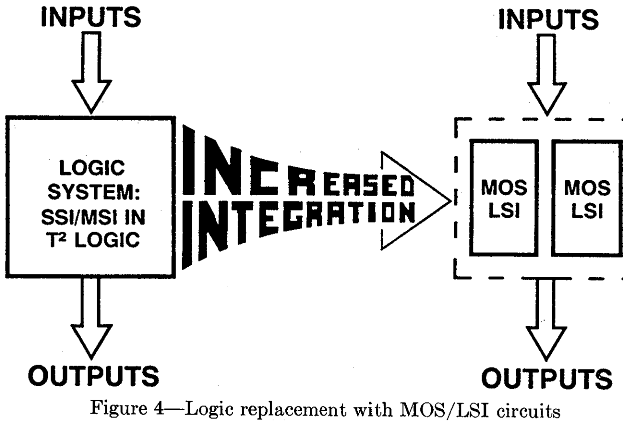


Figure 4—Logic replacement with MOS/LSI circuits

TABLE I—Random Logic vs Programmed Logic

RANDOM	PROGRAMMED
Special Purpose Components	General Purpose Components
Low Volume	High Volume
Dedicated Hardware	Dedicated Software
Moderate Speed	Slower Speed
Difficult to Modify	Easily Modified
Difficult to Expand	Easily Expanded

pressed to distinguish between a component and a system. In the past, IC manufacturers have built components. With the introduction of the microprocessor comes the need to treat a component as though it were a system, which in effect it is. Therefore, the IC house will have to supply far more support than is usually required for a component. This support commitment is a significant factor in the successful manufacture of integrated processors. Those suppliers who support their product only at the component level will require a great deal of assistance from the intermediary system consultant companies that are beginning to appear on the scene. Without such assistance, a component-level support effort is doomed to failure in the general integrated processor market.

However, at the other end of the spectrum, the component supplier becomes an end-user system supplier who competes with his former customers. The component manufacturer who wishes to properly support his integrated processor while not competing with his customers must seek an intermediate support commitment. This level might include:

- components—CPU's, memories, and interface
- software—assemblers, simulators, utility programs
- board-level and system-level prototyping aids
- training aids—seminars, workshops, user's libraries

To avoid the specialization quandry discussed earlier, the IP suppliers are usually introducing general purpose integrated processors first. The devices have features which allow them to handle the wide range of applications indicated in Figure 6. The price paid for flexibility is usually speed. Thus, where higher speed is needed and increased specialization is economically justified, special purpose processors will be built.

One measure of merit of an integrated processor is its ease of use. Ease of use applies to three categories: hardware, software, and support. Ultimately, ease of use translates into

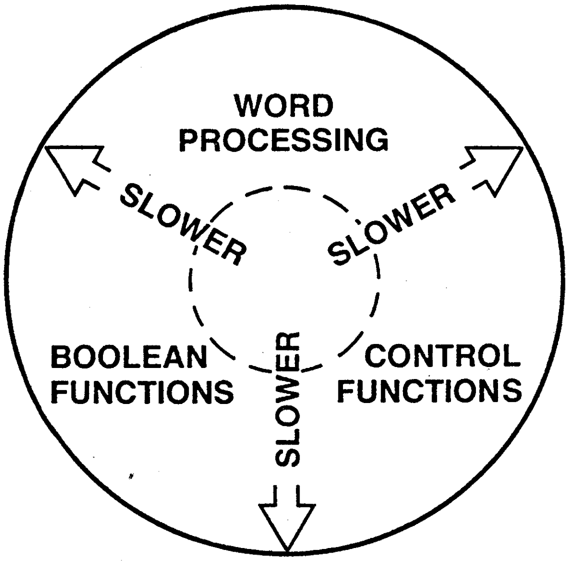


Figure 6—Microprocessor applications

TABLE II—2650 Features

Single Chip	Single +5 Volt Supply
Fixed Instruction Set	TTL/IO
Parallel 8 Bit	Static Logic
40 Pin Dip	Single Phase Clock
N-Channel 1^2 Si Gate	TTL Level Clock
32K Byte Range	Eight Address Modes
Address, Data Busses	Vectored Interrupt
75 Instructions	7 General Purpose Registers
1, 2, 3 Byte Instructions	Return Address Stack On Chip
Instruction Times 10us	Program Status Word

TABLE IV—Interface Signal Summary

PINS	TYPE	SIGNAL
2	Power	+5V, Ground
1	Input	Clock
1	Input	Sense
1	Input	Pause
1	Input	Reset
1	Input	Address Enable
1	Input	Data Bus Enable
1	Input	Interrupt Request
1	Input	Operation Acknowledge
8	I/O	Data Bus
13	Output	Address
1	Output	Address—Data/Control
1	Output	Address—Long/Short
1	Output	Memory/I-O
1	Output	Read/Write
1	Output	Operation Request
1	Output	Write Pulse
1	Output	Interrupt Acknowledge
1	Output	Run/Wait
1	Output	Flag

TABLE V—SIGNETICS 2650—Manufacturer Supplied Chips

LSI PRODUCTS	DESCRIPTION
2650	NMOS 8-bit Static Microprocessor
2602	NMOS 1024x1 Static RAM
2604	NMOS 4096x1 Dynamic RAM
2606	NMOS 256x4 Static RAM
2608	NMOS 1024x8 Static ROM
8T31	STTL 8-bit Bidirectional Port
82S123	STTL 32x8 Tristate PROM
82S129	STTL 265x4 Tristate PROM
82S131	STTL 512x4 Tristate PROM
82S115	STTL 512x8 Tristate PROM
LOGIC FAMILIES	
8T00	Interface SSI/MSI/LSI
7400, 8200	TTL SSI/MSI/Memory
82S00	STTL Memory
74LS00	Low Power STTL SSIMSI
4000	CMOS SSI/MSI

TABLE III—Instruction Set Summary

Load/Store Instructions			Length (bytes)
LODZ	r	Load Register Zero	1
LODI,r	v	Load Immediate	2
LODR,r	(*)a	Load Relative	2
LODA,r	(*)a(,X)	Load Absolute	3
STRZ	r	Store Register Zero	1
STRR,r	(*)a	Store Relative	2
STRA,r	(*)a(,X)	Store Absolute	3

Arithmetic Instructions			
ADDZ	r	Add to Register Zero	1
ADDI,r	v	Add Immediate	2
ADDR,r	(*)a	Add Relative	2
ADDA,r	(*)a(,X)	Add Absolute	3
SUBZ	r	Subtract from Register Zero	1
SUBI,r	v	Subtract Immediate	2
SUBR,r	(*)a	Subtract Relative	2
SUBA,r	(*)a(,X)	Subtract Absolute	3

Logical Instructions			
ANDZ	r	And to Register Zero	1
ANDI,r	v	And Immediate	2
ANDR,r	(*)a	And Relative	2
ANDA,r	(*)a(,X)	And Absolute	3
IORZ	r	Inclusive Or to Register Zero	1
IORI,r	v	Inclusive Or Immediate	2
IORR,r	(*)a	Inclusive Or Relative	2
IORA,r	(*)a(,X)	Inclusive Or Absolute	3
EORZ	r	Exclusive Or to Register Zero	1
EORI,r	v	Exclusive Or Immediate	2
EORR,r	(*)a	Exclusive Or Relative	2
EORA,r	(*)a(,X)	Exclusive Or Absolute	3

Comparison Instructions			
COMZ	r	Compare to Register Zero	1
COMI,r	v	Compare Immediate	2
COMR,r	(*)a	Compare Relative	2
COMA,r	(*)a(,X)	Compare Absolute	3

Rotate Instructions			
RRR,r		Rotate Register Right	1
RRL,r		Rotate Register Left	1

Branch Instructions			Length (bytes)
BCTR,v	(*)a	Branch On Cond. True Rel.	2
BCFR,v	(*)a	Branch On Cond. False Rel.	2
BCTA,v	(*)a	Branch On Cond. True Abso.	3
BCFA,v	(*)a	Branch On Cond. False Abso.	3
BRNR,r	(*)a	Branch On Reg. Non-Zero Rel.	2
BRNA,r	(*)a	Branch On Reg. Non-Zero Abso.	3
BIRR,r	(*)a	Branch On Incre. Reg. Rel.	2
BIRA,r	(*)a	Branch On Incre. Reg. Abso.	3
BDRR,r	(*)a	Branch On Decre. Reg. Rel.	2
BXA	(*)a,x	Branch Index. Abso., Uncond.	3
ZBRR	(*)a	Zero Branch Rel., Uncond.	2
BDRA,r	(*)a	Branch On Decre. Reg. Abso.	3

Subroutine Branch/Return Instructions			
BSTR,v	(*)a	Br. Subrou. Cond. True, Rel.	2
BSFR,v	(*)a	Br. Subrou. Cond. False, Rel.	2
BSTA,v	(*)a	Br. Subrou. Cond. True, Abso.	3
BSFA,v	(*)a	Br. Subrou. Cond. False, Abso.	3
BSNR,r	(*)a	Br. Subrou. Non-Zero Reg. Rel.	2
BSNA,r	(*)a	Br. Subrou. Non-Zero Reg. Abso.	3
BSXA	(*)a,x	Br. Subrou., Indexed, Uncond.	3
RETC,v		Ret. From Subrou., Cond.	1
RETE,v		Ret. Sub. and Enab. Intr., Cond.	1
ZBSR	(*)a	Zero Br. To Sub. Rel., Uncond.	2

Program Status Instructions			
LPSU		Load Program Status, Upper	1
LPSL		Load Program Status, Lower	1
SPSU		Store Program Status, Upper	1
SPSL		Store Program Status, Lower	1
CPSU	v	Clear Pro. Stat., Upper, Mask.	2
CPSL	v	Clear Pro. Stat., Lower, Mask.	2
PPSU	v	Preset Pro. Sta., Upper, Mask.	2
PPSL	v	Preset Pro. Sta., Lower, Mask.	2
TPSU	v	Test Pro. Status, Upper, Mask.	2
TPSL	v	Test Pro. Status, Lower, Mask.	2

Input/Output Instructions			
WRTD,r		Write Data	1
REDD,r		Read Data	1
WRTC,r		Write Control	1
REDC,r		Read Control	1
WRTE,r	v	Write Extended	2
REDE,r	v	Read Extended	2

Miscellaneous Instructions			
HALT		Halt, Enter Wait State	1
DAR,r		Decimal Adjust Register	1
TMI,r	v	Test Under Mask Immediate	2
NOP		No Operation	1

Explanation of Symbols

( )—indicates option
r—register expression
v—value expression
*—indirect indicator
a—address expression
x—index register expression
X—index register expression with optional auto-increment or auto-decrement

development and system economy factors. Since system cost will largely depend on interface and memory costs rather than IP cost, IP's that have powerful instruction sets and minimize the interface circuitry will significantly reduce the system cost. In addition to reducing the price of the memory, IP's with good coding efficiency will improve throughput performance.

The general purpose CPU's being offered on the market today generally have several features in common:

- single chip construction
- fixed instruction sets
- eight-bit parallel data
- MOS/LSI technology
- 40-pin dual in-line packaging

Beyond these features, however, each supplier differs in his approach to the requirements of the introduction to the 2650.

The following discussion briefly describes the 2650 microprocessor manufactured by Signetics Corporation. The 2650 is a complete general purpose integrated processor on one monolithic integrated circuit. Table II summarizes the features of the 2650. System objectives achieved by the 2650 include:

(1) Minimizing the amount of memory needed to perform a given function.
(2) Minimizing the amount of interface logic required to implement an IP-based system.
(3) Providing an easily learned instruction set that is based on already-existing computer architectures.

Table III contains a summary of the instruction set with addressing modes which significantly reduce the size of the memory needed to perform a given function. Table IV lists the interface signals available. These signals reduce the amount of external interface circuitry needed to operate the processor. All inputs and outputs of the 2650 are TTL compatible. A list of support circuits offered by Signetics is shown in Table V.

In summary, the 2650 offers those features needed for economical utilization of a general purpose IP in the logic replacement marketplace. The evolution of the integrated processor has combined the IC technology with existing system concepts to offer an attractive alternative for logic system design.

Keeping pace with a single-chip 16-bit microprocessor

by ALAN J. WEISSBERGER
National Semiconductor Corporation
Santa Clara, California

INTRODUCTION

The emphasis in contemporary microprocessor development has been on 8-bit word lengths. Unfortunately, for many applications, the 8-bit microprocessor cannot provide the required accuracy, throughput, programming ease, or flexibility. The multichip 16-bit processor has been cost effective in many of these applications, but has provided unused flexibility or speed (at extra cost) in others. National Semiconductor has developed a single-chip 16-bit microprocessor, the Processing and Control Element (PACE), to provide the benefits of a 16-bit CPU with greater simplicity than the multi-chip design. The benefits accrue from integrating the functions of not only the multi-chip CPU, but also most of the functions that were previously implemented with TTL devices.

In addition, a group of compatible microcomputer chips has been developed to augment the basic processor. A complete microcomputer system, with 1024 words (16,384 bits) of read-only program storage, clocks, buffers and one 16-bit or two 8-bit peripherals is shown in Figure 1. Table I lists features and benefits of this microprocessor.

ARCHITECTURE

The PACE microprocessor, shown in Figure 2, provides 16-bit parallel data-processing capability in a 40-pin package. Functionally, the processor can be segmented into six blocks: Data Storage, ALU, Status, Control, Interrupts, Input, and Output.[1]

Four accumulators, two temporary registers, a program counter, and a 10-word Last-In/First-Out Stack (LIFO) provide ample storage for data manipulation, address formation, and arithmetic computations. Two of the accumulators (AC0, AC1) are principal working registers, while the two others (AC2, AC3) may be used as index registers or auxiliary working registers. The LIFO stack is used primarily to save the program counter during subroutine execution or interrupt servicing. It can also be used to store status information or data. External read/write memory may be used as a stack extension by provision of stack-full and stack-empty interrupts, allowing implementation of a simple stack-service routine.

Arithmetic Logic Unit (ALU) operations include AND, OR, XOR, complement, shift left, shift right, mask byte, and sign extend. Both binary and 4-digit BCD addition capability are provided, thus eliminating the program storage and execution time required to perform BCD to binary conversion. A unique feature of the PACE ALU is the ability to operate on either 8- or 16-bit data, as specified by the programmer through the use of a status flag. This feature allows character-oriented and other 8-bit applications to be implemented and executed using an 8-bit peripheral data bus and read-write memory, while address formation and instruction storage are implemented in the more-effective 16-bit data length.

All status and control bits for PACE are provided in a single Status flag register, whose contents may be loaded from or to any accumulator or the stack. This allows convenient testing, masking and storage of status. In addition, a number of status bits may be tested directly by the conditional branch instruction, and any bit may be individually set or reset. The byte flag is used to specify an 8-bit data length for data processing instructions, while arithmetic operations for address formation remain at the 16-bit data length. In the 8-bit data mode, modifications of the carry, overflow, and link flags are based on the 8 least significant data bits only. Four flags (bits 11-14) that may be assigned functions by the programmer are provided. These flags drive output pins and may be used to directly control system functions or as software status flags.

Six levels of prioritized vector interrupts are available. This allows automatic identification of an interrupting device's level by trapping to a dedicated location in an interrupt pointer table. The pointer specifies the starting address of the interrupt service routine for that particular level. All devices on a given level can be enabled or disabled as a group, independent of other interrupt levels. This permits a fast responding peripheral device on a high level to interrupt a slower peripheral device on a lower level. An individual interrupt enable is provided in the status register for each level (IE1 to IE5), and a master interrupt enable (IEN) is provided for all five lower priority levels as a group. The level-zero interrupt is an exception to this procedure. It is the highest priority interrupt in the system and cannot be locked out by the master interrupt enable. This interrupt level is typically used by the control panel, which can then interrupt the

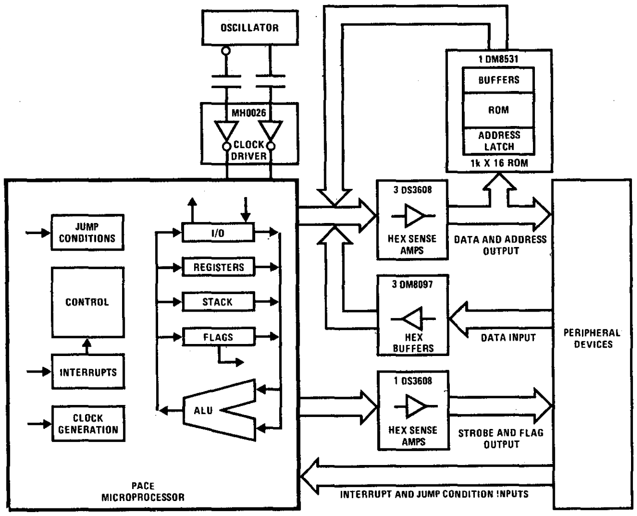


Figure 1—PACE system with 1k×16 ROM—A complete data processing system can be built with PACE, a clock driver and input and output buffers. The control program is stored in a one-chip ROM

application program without affecting system status. It could also be used as an indication of a catastrophic condition such as a power failure. In this case the processor would save its internal registers in a non-volatile or battery supplied memory and halt execution in an orderly fashion.

The minimal package count required to implement a microprocessor system using PACE and its support chips may be important in some applications independent of the associated lower cost. Hand-held or portable equipment may have physical constraints that can only be met by a processor component family of parts. Low power dissipation may also be important in some applications, and the use of a MOS microprocessor with CMOS or lower power TTL support chips may be required.

Some applications that might benefit from the small size, weight, and power requirement of the PACE microcomputer system include remote sensing systems, weather-monitoring stations, and natural-gas pipelines. In each case, a minimum PACE microcomputer system could be installed at an unmanned site. Information could be sensed, collected, and processed locally before being sent to a central computer or recorded on a cassette. Local control and preprocessing reduces data transmission costs because only tested and verified data is sent.[4] These unmanned microprocessor-based systems could also run calibration and diagnostic tests of the remote instrumentation to determine whether or not it is functioning properly.

The ability to operate on either 8- or 16-bit data can be a great advantage in terminals and communication processors. Eight-bit characters can be extracted and processed in the 8-bit mode of operation without packing and unpacking overhead software. Line monitoring, statistical tabulations and error control may be implemented using 16 bits. The PACE CPU can be conveniently interfaced to a byte-oriented peripheral (CRT) and to equipment that has a data length exceeding 8 bits (card reader).

Command outputs and external status inputs are implemented very efficiently using the PACE CPU. The flag outputs can be utilized for control functions, such as start reader, rewind, and others in a tape controller. Similarly, the user jump conditions can be used to sense system status conditions, such as end of tape or inter-record gap. A flag and jump condition can be used together as a serial I/O port, eliminating the hardware required to interface to the data bus and to decode the device address. Several flags and jump condition inputs can be used to provide a keyboard scanning function, modem control, or character synchronization in a smart terminal.

The PACE interrupt system can save considerable hardware and software in applications having several interrupts. The on-chip priority logic and vectored branch to the interrupt routine save logic required external to other microprocessors to resolve priority and jam an address vector onto the data bus, or the program storage and execution time required for the alternative scheme of sequentially polling the interrupt status of all devices. Interrupts are essential in applications where alarm conditions or transient conditions must be serviced immediately, such as automobile, process or machine tool control, or plant monitoring. They are useful in many other systems to eliminate the program overhead required to scan asynchronous system inputs, such as a controller for multiple terminals or an intersection traffic-light controller.

The ability to add BCD data eliminates execution time and program storage overhead required to convert BCD to binary data. This is useful in BCD-oriented applications, such as display controllers, electronic cash registers, billing systems, accounting machines, navigation aids, and industrial controllers and test systems.

The compatibility of PACE with the microprogrammable IMP-16 is beneficial in applications where the IMP-16 could serve as a host processor with the PACE being used as a lower-level processor, such as an automated assembly line. Applications where a microprocessor controlled product is available in several models may use the IMP-16 for the more-sophisticated models and the PACE for the less demanding tasks, allowing common software and peripheral interfaces.

Data transfers between PACE and external memories or peripheral devices take place over the 16 data lines (D00-D15); are synchronized by 4 control signals (NADS, IDS, ODS, and EXTEND); and use common instructions. This

TABLE I—PACE Features

• 16-bit instruction word	Addressing flexibility, speed
• 8- or 16-bit data word	Wide application
• 45 instructions	Efficient programming
• Common memory and peripheral addressing	Powerful I/O instructions
• Shares instructions with National's IMP-16	Allows software compatibility
• 4 general purpose accumulators	Reduces memory data transfers
• 10-word stack	Interrupt processing/data storage
• 6 vectored priority interrupt levels	Simplifies interrupt service and hardware
• Programmer accessible status register	May be preserved, tested, or modified
• Typical 10μsec instruction execution	High speed
• Can utilize DM8531 1k-by-16 ROM	Single memory package
• Single-phase true and complement clock	Minimum external components

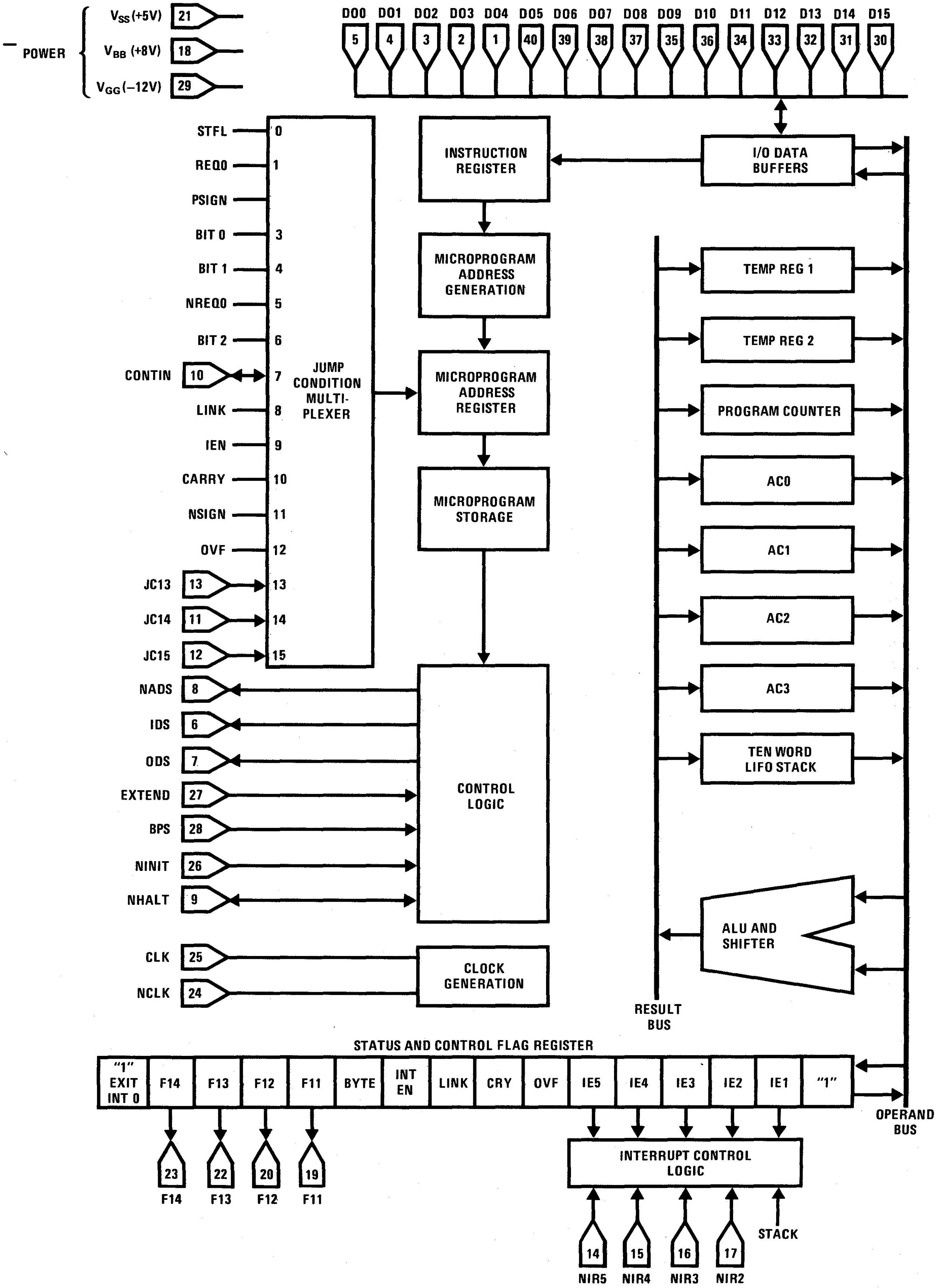


Figure 2—PACE detailed block diagram

```
CONST:  .WORD X'FFFF      ; CONSTANT FOR DOUBLE PREC. ADD
START:  LI    R1, 0       ; CLEAR RESULT REGISTER
        LI    R3, 16      ; LOOP COUNT TO AC3
        CAI   R0, 0       ; COMPLEMENT MULTIPLIER
LOOP:   RADD  R1, R1      ; SHIFT RESULT LEFT INTO CARRY
        RADC  R0,R0       ; SHIFT CARRY INTO MULTIPLIER
                          ; AND MULTIPLIER INTO CARRY
        BOC CARRY, TEST   ; TEST FOR ADD
        RADD  R2, R1      ; ADD MULTIPLICAND TO RESULT
        SUBB  R0, CONST   ; ADD CARRY TO H.O. RESULT
TEST:   AISZ  R3, -1      ; DECREMENT LOOP COUNT
        JMP LOOP          ; REPEAT LOOP
```

Figure 3—Multiply routine

unified bus architecture is in contrast with many other microprocessor or minicomputers that have one instruction type (I/O class) for communication with peripheral devices and another instruction type (memory-reference class) for communication with memories. The advantage of the approach used by PACE is that all memory-reference instructions are available for communication with peripherals. For example, the DSZ (Decrement and Skip if Zero) instruction can be used to decrement and test a peripheral device register; the SKAZ (Skip if And is Zero) instruction can be used to test the contents of a status register; LD (Load) and ST (Store) instructions may be used for simple data transfers. This technique can improve throughput and simplify programming.

Data transfer operations are initiated by an address data strobe (NADS), which gates the address to the memory or peripheral. An input or output data strobe (IDS or ODS) follows on the next clock cycle. The appropriate strobe is used to gate the data into or out of the processor. The memory device shown in Figure 1 provides address latches on the chip. Two 8-bit bidirectional TRI-STATE data latches may be provided for the peripheral(s). The EXTEND input allows the I/O cycle time to be extended by multiples of the clock cycle to adapt to a variety of memory and peripheral devices or for DMA bus interfacing. Further functional details are provided in References 2 and 3.

Programming

An 8-bit processor must manipulate multiple registers to form 16-bit addresses, make several memory accesses to fetch multi-byte instructions or 16-bit data and use double precision arithmetic routines to obtain accuracy greater than two decimal places. A 16-bit processor does not suffer from these limitations so that faster, shorter and simpler programs may be written. This is clearly evident in minicomputers where the 16-bit word length is standard.

The sample program of Figure 3 illustrates the efficiency of PACE in data processing applications. The complete instruction set, divided into eight instruction classes, is listed in Table II.

The program multiplies the 16-bit value in AC2 (multiplicand) by the 16-bit value in AC1 (multiplier) and provides a 32-bit result in AC0 (high order) and AC1 (low order). Worst case execution time is under one millisecond.

UNIQUE FEATURES

Many of the features of the PACE microprocessor prove beneficial for a wide range of applications, while some provide direct benefits in certain classes of application. The 16-bit instruction and address word lengths and multiple accumulator architecture make programming easier and more efficient. Instructions and operands are fetched in single memory cycles rather than the multiple memory references required for byte-oriented data or instructions. This enhances system throughput and improves program execution times. Program storage requirements and development cost reductions sometimes allow more hardware functions to be implemented in software, reducing system cost and making more of the system reconfigurable by software modification.

Certain functions implemented on the chip simplify interfacing by minimizing the number of external components for a microcomputer system.

- Internal Clock generation from the true and complement clock inputs eliminates the need for a complicated timing generator.
- On-chip output buffers drive sense amplifiers with TRI STATE capability. This reduces power dissipation and chip size while improving speed.
- Interrupt control logic on the chip improves interrupt response time and saves 15-20 TTL packages that would ordinarily be required for the equivalent function.
- The jump condition multiplexer, status and control flag register are internal functions for sensing inputs and providing outputs directly to the user.

APPLICATION

The ability to efficiently operate on 8 or 16 bit data and perform binary or BCD arithmetic enables PACE to act as a controller or data processor in a complex system environment. In many cases a minicomputer or multiple dedicated microprocessors could be replaced with substantial savings in cost and complexity.

To illustrate the flexibility and power of the PACE microprocessor an application example has been developed. The Plant Security Monitoring System (PSMS), shown in Figure 4, acts as a watchdog by monitoring and in some instances controlling a plant's operation. One PACE CPU acts as a data acquisition/alarm scanner while another PACE CPU is utilized as a central control/acknowledgment terminal. The functions monitored are plant power (peak demand, total consumption, outage) and environmental quality (air contaminants,

TABLE II—PACE Instruction Summary

	Mnemonic	Meaning	Operation	Assembler Format		Instruction Format			
1.	**Branch Instructions**								
	BOC	Branch On Condition	(PC) ← (PC) + disp if cc true	BOC	cc,disp	0 1 0 0	cc	disp	
	JMP	Jump	(PC) ← EA	JMP	disp (xr)	0 0 0 1 1 0	xr	disp	
	JMP@	Jump Indirect	(PC) ← (EA)	JMP	@disp (xr)	1 0 0 1 1 0			
	JSR	Jump To Subroutine	(STK) ← (PC), (PC) ← EA	JSR	disp (xr)	0 0 0 1 0 1			
	JSR@	Jump To Subroutine Indirect	(STK) ← (PC), (PC) ← (EA)	JSR	@disp (xr)	1 0 0 1 0 1			
	RTS	Return from Subroutine	(PC) ← (STK) + disp	RTS	disp	1 0 0 0 0 0	0 0	disp	
	RTI	Return from Interrupt	(PC) ← (STK) + disp, IEN = 1	RTI	disp	0 1 1 1 1 1			
2.	**Skip Instructions**								
	SKNE	Skip if Not Equal	If (ACr) ≠ (EA), (PC) ← (PC) + 1	SKNE	r,disp (xr)	1 1 1 1 r	xr	disp	
	SKG	Skip if Greater	If (AC0) > (EA), (PC) ← (PC) + 1	SKG	0,disp (xr)	1 0 0 1 1 1			
	SKAZ	Skip if And is Zero	If [(AC0) ∧ (EA)] = 0, (PC) ← (PC) + 1	SKAZ	0,disp (xr)	1 0 1 1 1 0			
	ISZ	Increment and Skip if Zero	(EA) ← (EA) + 1, if (EA) = 0, (PC) ← (PC) + 1	ISZ	disp (xr)	1 0 0 0 1 1			
	DSZ	Decrement and Skip if Zero	(EA) ← (EA) – 1, if (EA) = 0, (PC) ← (PC) + 1	DSZ	disp (xr)	1 0 1 0 1 1			
	AISZ	Add Immediate, Skip if Zero	(ACr) ← (ACr) + disp, if (ACr) = 0, (PC) ← (PC) + 1	AISZ	r,disp	0 1 1 1 1 0	r		
3.	**Memory Data Transfer Instructions**								
	LD	Load	(ACr) ← (EA)	LD	r,disp (xr)	1 1 0 0 r	xr	disp	
	LD@	Load Indirect	(AC0) ← ((EA))	LD	0,@disp (xr)	1 0 1 0 0 0			
	ST	Store	(EA) ← (ACr)	ST	r,disp (xr)	1 1 0 1 r			
	ST@	Store Indirect	((EA)) ← (AC0)	ST	0,@disp (xr)	1 0 1 1 0 0			
	LSEX	Load With Sign Extended	(AC0) ← (EA) bit 7 extended	LSEX	0,disp (xr)	1 0 1 1 1 1			
4.	**Memory Data Operate Instructions**								
	AND	And	(AC0) ← (AC0) ∧ (EA)	AND	0,disp (xr)	1 0 1 0 1 0	xr	disp	
	OR	Or	(AC0) ← (AC0) ∨ (EA)	OR	0,disp (xr)	1 0 1 0 0 1			
	ADD	Add	(ACr) ← (ACr) + (EA), OV, CY	ADD	r,disp (xr)	1 1 1 0 r			
	SUBB	Subtract with Borrow	(AC0) ← (AC0) + ~ (EA) + (CY), OV, CY	SUBB	0,disp (xr)	1 0 0 1 0 0			
	DECA	Decimal Add	(AC0) ← (AC0) $+_{10}$ (EA) $+_{10}$ (CY), OV, CY	DECA	0,disp (xr)	1 0 0 0 1 0			
5.	**Register Data Transfer Instructions**								
	LI	Load Immediate	(ACr) ← disp	LI	r,disp	0 1 0 1 0 0	r	disp	
	RCPY	Register Copy	(ACdr) ← (ACsr)	RCPY	sr,dr	0 1 0 1 1 1	dr	sr	not used
	RXCH	Register Exchange	(ACdr) ← (ACsr), (ACsr) ← (ACdr)	RXCH	sr,dr	0 1 1 0 1 1			
	XCHRS	Exchange Register and Stack	(STK) ← (ACr), (ACr) ← (STK)	XCHRS	r	0 0 0 1 1 1	r	not used	
	CFR	Copy Flags Into Register	(ACr) ← (FR)	CFR	r	0 0 0 0 0 1			
	CRF	Copy Register Into Flags	(FR) ← (ACr)	CRF	r	0 0 0 0 1 0			
	PUSH	Push Register Onto Stack	(STK) ← (ACr)	PUSH	r	0 1 1 0 0 0			
	PULL	Pull Stack Into Register	(ACr) ← (STK)	PULL	r	0 1 1 0 0 1			
	PUSHF	Push Flags Onto Stack	(STK) ← (FR)	PUSHF		0 0 0 0 1 1	not used		
	PULLF	Pull Stack Into Flags	(FR) ← (STK)	PULLF		0 0 0 1 0 0			
6.	**Register Data Operate Instructions**								
	RADD	Register Add	(ACdr) ← (ACdr) + (ACsr), OV, CY	RADD	sr,dr	0 1 1 0 1 0	dr	sr	not used
	RADC	Register Add With Carry	(ACdr) ← (ACdr) + (ACsr) + (CY), OV, CY	RADC	sr,dr	0 1 1 1 0 1			
	RAND	Register And	(ACdr) ← (ACdr) ∧ (ACsr)	RAND	sr,dr	0 1 0 1 0 1			
	RXOR	Register Exclusive OR	(ACdr) ← (ACdr) ⊻ (ACsr)	RXOR	sr,dr	0 1 0 1 1 0			
	CAI	Complement and Add Immediate	(ACr) ← ~ (ACr) + disp	CAI	r,disp	0 1 1 1 0 0	r	disp	
7.	**Shift And Rotate Instructions**								
	SHL	Shift Left	(ACr) ← (ACr) shifted left n places, w/wo link	SHL	r,n,ℓ	0 0 1 0 1 0	r	n	ℓ
	SHR	Shift Right	(ACr) ← (ACr) shifted right n places, w/wo link	SHR	r,n,ℓ	0 0 1 0 1 1			
	ROL	Rotate Left	(ACr) ← (ACr) rotated left n places, w/wo link	ROL	r,n,ℓ	0 0 1 0 0 0			
	ROR	Rotate Right	(ACr) ← (ACr) rotated right n places, w/wo link	ROR	r,n,ℓ	0 0 1 0 0 1			
8.	**Miscellaneous Instructions**								
	HALT	Halt	Halt	HALT		0 0 0 0 0 0	not used		
	SFLG	Set Flag	$(FR)_{fc}$ ← 1	SFLG	fc	0 0 1 1	fc	1	not used
	PFLG	Pulse Flag	$(FR)_{fc}$ ← 1, $(FR)_{fc}$ ← 0	PFLG	fc	0 0 1 1	fc	0	

temperature, air flow). Various transducers, thermocouples and sensing devices measure the required analog variables and provide inputs to an analog multiplexer. PACE scans these input points at operator selected time intervals by supplying a point address to the analog multiplexer and starting the Analog to Digital (A/D) converter. When the conversion is complete the data are read, processed, and checked against alarm limits. Critical deviations from normal operating conditions are detected and alarms are sent to the control/acknowledgment terminal. The PACE CPU at the terminal formats and routes the alarm data to an operators display panel. The operator

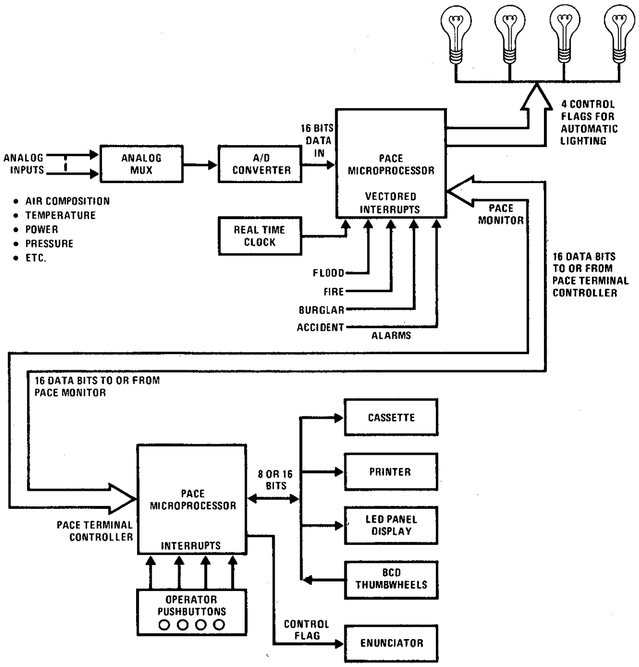


Figure 4—(Application example). PACE as a plant monitor and terminal controller

on duty observes the detected alarm and takes the necessary steps to correct the problem. Some alarms can be detected directly by limit switches, continuity breakage or by manually pressing a button. Examples include floods, fire, burglary, or accident alarms. These "crisis" conditions require immediate attention and would therefore be implanted as prioritized vector interrupts in the PACE monitor. Fast response and immediate operator notification are guaranteed by the sounding of an annunciator horn at the control terminal.

In addition to the above monitoring chores, one or more simple control functions could be provided. For automatic light control, shown in the example, a real time clock generates interrupt signals at fixed preset intervals. The processor recognizes the time of each interrupt and, if appropriate, dims the lights or turns them on or off. Light control commands are facilitated through the four user flags on the chip. This function would conserve energy by providing efficient allocation of electricity. Temperature control of the building by regulating heaters and air conditioners is another possible function that might be implemented as a dedicated application.

The operator at the central control terminal can select various status conditions to be displayed or he can change alarm limits through a set of BCD thumbwheels. Pushbuttons are used as interrupts to get the processors attention. A tape cassette or printer might be provided for record keeping or hard copy outputs. The PACE terminal controller works primarily with 8-bit character data for the supporting peripherals, but it can process 16-bit data from the thumbwheels or the monitor controller. This unique feature (selectable 8 or 16-bit data processing) can be used to efficiently adapt PACE to the function required. Auxiliary functions like trend analysis or signal averaging, could be provided by either PACE microprocessor, depending on the respective data load. Note also that binary and BCD data (thumbwheels and LED's) are processed directly.

CONCLUSION

The PSMS is a solution to a complex problem that is common to all industries. This application offers PACE as a system solution to a multitude of specific tasks. These tasks would ordinarily be done manually, with reduced efficiency, or electrically, with increased complexity and cost. The interfacing simplicity, benefits and low cost of LSI, and the convenience of working with 16-bits promise to make PACE a universal tool in many existing and new applications for microprocessors.

ACKNOWLEDGMENTS

The author would like to thank George Reyling and Gary Miller for their valuable contributions in preparation of this paper.

REFERENCES

1. Reyling, George F., "*Single Chip Microprocessor Employs Minicomputer Word Length,*" *Electronics,* December 26, 1974, pp. 87-93.
2. *PACE Data Sheet,* IPC-16A/500D, National Semiconductor Corp.
3. *PACE Users Manual,* National Semiconductor Corp.
4. Weissberger, Alan J., "*Microprocessor as Intelligent Remote Controllers,*" WESCON 74, Session 23.

Tools and techniques of microprocessor data transfer

by GARY SAWYER
Motorola
Phoenix, Arizona

INTRODUCTION

This paper discusses I/O transfer from four different viewpoints. The first, "I/O Data Transfer Techniques", contrasts the two basic methods of I/O data movement. Having discussed the movement of data, "I/O control" focuses in on how the data movement is governed. "I/O Interface Hardware" takes a closer look at specific hardware that may be used to interface to the MPU. Finally, "An Example of I/O Transfer" concludes the discussion showing the software and hardware required in an actual transfer.

I/O DATA TRANSFER TECHNIQUES

The I/O capability of a microprocessor is a key standard of measure. As microprocessors mature, more techniques are becoming available at better throughput speeds. It, therefore, becomes useful to put these techniques in perspective by categorizing how data is shipped through the microprocessor system. The first, and most commonly used, technique is to ship data through the microprocessor (MPU) wherein the MPU acts as a data funnel to the outside world. The second technique, direct memory access (DMA), transfers data directly between memory and the outside world circumventing the MPU. Following is a description of each with a discussion of associated vices and virtues.

The first technique considered moves data through the system via the MPU under program control. The MPU then becomes the focus for data movement between the peripheral and memory. Figure 1 shows, at a block diagram level, where data is moving during an I/O transfer. It is comprised of four basic hardware blocks: the MPU, memory, I/O interface, and a peripheral. If, for example, data is to be transferred from the peripheral to the system memory, the first link in the chain is the I/O interface. The characteristics of this interface are, as expected, a function of (1) the data and control requirements of the peripheral, and (2) the processor used in the system. A discussion of I/O interface hardware is an important topic and is treated in the section "Interfacing to the Microprocessor". Once data has been shipped to the interface, the MPU reads the data from the interface device. The MPU may now complete the I/O transfer by storing the data in the desired memory location. If, on the other hand, data is transferred from memory to peripheral, the sequence of events is reversed; the MPU reads the data from memory and stores data into the I/O interface for transfer to the peripheral. Data is, therefore, transferred from block to block under program control.

Microprocessors available today use two general classes of instructions to move data: One, use of an Input or Output instruction or, two, a memory reference instruction. When, for example, the MPU is given an Input or Output instruction, the microprocessor will issue control signals and address the desired I/O interface device. The second class of instruction to access I/O is memory reference. Here the I/O interface is assigned a memory address and is accessed during any instruction that specifies the defined I/O interface.

Direct memory access is the second alternative to I/O data transfer. The MPU is circumvented and the data moves directly between memory and peripheral. Figure 2 is a representation of how data will move through the system using DMA techniques. Here, the microprocessor is off the bus and the DMA interface transfers directly to/from the memory. This requires the DMA interface to (1) override the MPU operation causing it to go into an off (high impedance) state, and (2) generate memory address and control signals for the desired data transfer.

Comparison of the two techniques is largely a function of speed and hardware. DMA, for example, will generally require more hardware due to the additional control tasks. On the other side of the coin, DMA is consistently the faster of the two transfers. Here, the data transfer is normally limited only by the cycle time of the memory. This feature becomes valuable when the MPU is not fast enough to handle transfer under program control. When funneling data through the MPU, the I/O interface is straightforward, but the transfer rate is now a function of instruction execution time. In this case the MPU becomes the limiting factor. The choice, as usual, is in the hands of the designer. He may require a fast DMA channel at the expense of hardware, or he may simplify the interface transferring the work load to software.

I/O CONTROL

A major aspect of I/O data transfer is that of control. Where is the I/O transfer initiated? How long does it continue? Which peripheral is to gain access to the system? These questions are recurring in virtually every micropro-

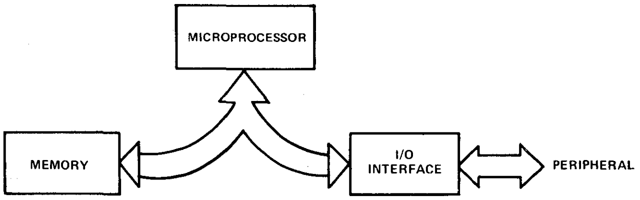


Figure 1—I/O Transfer through the MPU

cessor based system because of the dynamic nature of operation. The great strength of a microprocessor is the flexibility of the program. Programs are more than a list of instructions commanding a fixed sequence of operations. These programs can be written to adjust to external events, query peripherals for service or respond to hardware service requests. The list is unending, therefore, the question of control could be answered in a word—software. But hardware certainly has its place in control of I/O transfer. Following is a discussion of hardware/software control options in context with "MPU funneled" transfer and DMA transfer.

Consider first the control of I/O transfer when data is transferred through the MPU as shown in Figure 1. Here the I/O transfer may be initiated by either software or under interrupt control. Interrupts may be issued to the microprocessor from peripherals (either directly or via the I/O interface) to inform the MPU of a request for service. When service is granted, the MPU breaks away from the current program, saves its status, and begins an interrupt service routine. At this point, the MPU needs to determine the source of the interrupt (normally multiple interrupts outnumber the interrupt inputs available at the MPU). This can be done by either hardware or software. Hardware can be used to prioritize all interrupts into an 8-bit word for the MPU to read or software can poll each I/O interface to determine where the interrupt originated.

Control options are equally flexible for DMA types of transfer. Here, the software or the DMA peripheral can initiate an I/O transfer. The user may choose to specify the beginning address and length of transfer with software by loading "control" words into the DMA circuitry. At the other extreme, hardware may be the dominant force. The DMA hardware can conceivably initiate the transfer, generate addresses, define direction of data flow, and length of the DMA transfer. Here, again, the designer has hardware/software options to minimize his system while maximizing the I/O transfer.

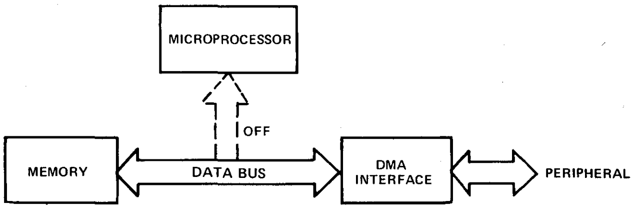


Figure 2—Direct memory access (DMA) I/O transfer

I/O INTERFACE HARDWARE

Transfer through the MPU

A common question asked by microprocessor users is how to interface between the MPU and the outside world. With regard to I/O transfer through the MPU, semiconductor houses are already ahead of the game. Companies such as Motorola, Rockwell and Intel have a host of interface devices available now with a promise of more. These devices run the gamut to anticipate the needs of the user. Some do little more than act as buffering latches. Others are customized to specific peripheral devices. A trend is developing toward a more sophisticated general purpose interface device whereby the interface may be programmed to assume a user defined personality.

A notable example of programmable interface devices available today is the MC6820 peripheral interface adapter (PIA) offered by Motorola. Figure 3 shows the PIA between the MPU and peripheral world. Notice that 16 data lines and 4 control lines are available to interface to a variety of peripherals. Each one of the data signals may be programmed to act as inputs or outputs in any combination. A user could, therefore, tie a number of input or output peripheral devices to a single PIA.

A unique feature of the PIA is the programmable control segment of the interface. The four control signals may be used by the designer to inform either the MPU or peripheral that an I/O transfer is occurring. If, for example, the peripheral transfers data to the MPU via a PIA, a pulse shipped in parallel to the control input will cause the PIA to generate an interrupt to the MPU. Should the MPU need to output data to the peripheral, the data may be stored into the PIA with a memory reference instruction. The PIA will save and transfer this data from the MPU data bus onto the peripheral data bus. The PIA can, for example, be programmed to then generate a control pulse to the peripheral informing the peripheral of new data. As a result, a data transfer to a peripheral with an equivalent "data present" pulse can be accomplished with a single STORE instruction by addressing the appropriate PIA.

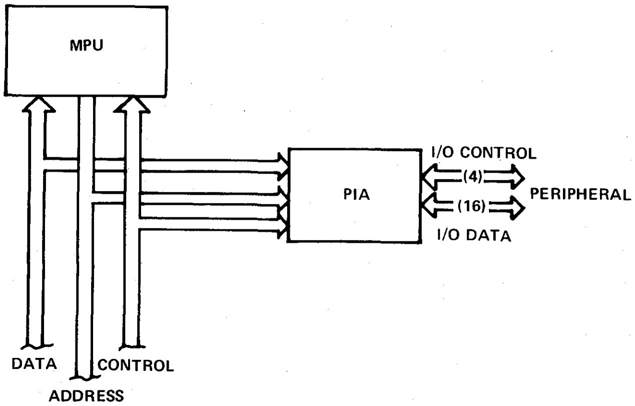


Figure 3—MPU parallel I/O interface

A closer view of the PIA will reveal how an interface device may be "programmed" by the microprocessor. Figure 4 shows six registers internal to the PIA divided into "A" and "B" sides. Each side of the PIA contains 8 data signals, 2 control signals, and three 8-bit registers. The user may, therefore, program peripheral data and control signals by loading words into respective PIA registers.

Peripheral data signals, PAO-PA7, may, for example, be programmed as inputs or outputs by loading the "A" data direction register. Each logic "Ø" of the data direction register will then define the respective peripheral data signal to be an input (the converse is true for a logic "1"). Similarly, the characteristics of the control signals, CA1 and CA2, are programmable by loading a word into the "A" control register. The control lines can be used as an input to the PIA to generate an MPU interrupt on either a rising or falling edge (if desired, the control input may also be masked off). As an output, the control signal may be programmed to act as a strobe or active level when moving data through the PIA. The same argument follows for the "B" side of the PIA. Loading data words into the "B" data direction register characterizes PBØ-PB7. Likewise, the "B" control register defines how CB1 and CB2 shall react.

DMA transfer

Design of a DMA interface requires close examination of the MPU in the system. In a typical direct memory access configuration the MPU and DMA share the system bus for control of the memory transfer. When the MPU is executing instructions the DMA circuitry is effectively off the bus. When the DMA transfer is initiated the MPU must be switched off the bus as the DMA interface switches on. The manner in which the MPU is removed from the bus becomes a major aspect of the DMA transfer.

A representative example of the mechanics of a DMA transfer is shown in Figure 5 using the Motorola MC6800 MPU. A number of techniques are available to control the MC6800 during a DMA transfer—here the $\overline{\text{Halt}}$ signal provides control over the MPU. The characteristics of the $\overline{\text{Halt}}$ line are such that, when low, the address, data and read/write signals go into a three-state condition at completion of the current instruction. When the $\overline{\text{Halt}}$ is recognized and the MPU in three-state, the MPU will bring the Bus Available signal high. The $\overline{\text{Halt}}$ and Bus Available signals of the MC6800 therefore become convenient DMA controls. Referring to Figure 5, the $\overline{\text{DMA Request}}$ initiates the transfer. Depending upon when the request is made with respect to the current instruction, the MPU will respond with a DMA Acknowledge signal within 2-14 μsec. When the Acknowledge is seen, the DMA interface is then free to take over the bus. At this time, the DMA circuitry has complete control over the memory transfer until the $\overline{\text{DMA Request}}$ returns to a high state. This requires the DMA circuitry to formulate desired addresses, to tie into the data bus, and generate the necessary control (R/W). When the DMA transfer is completed the interface switches off the bus and $\overline{\text{DMA Request}}$ returns to an inactive high state. With $\overline{\text{Halt}}$ inactive the MPU will then switch back onto the bus and continue program execution.

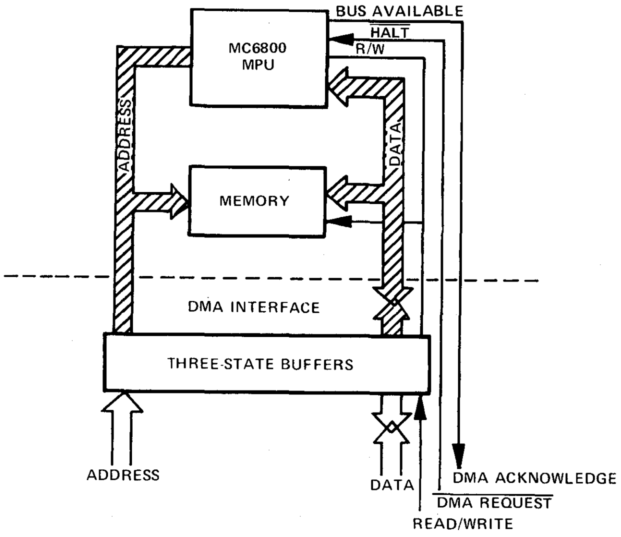


Figure 5—Direct memory access I/O transfer

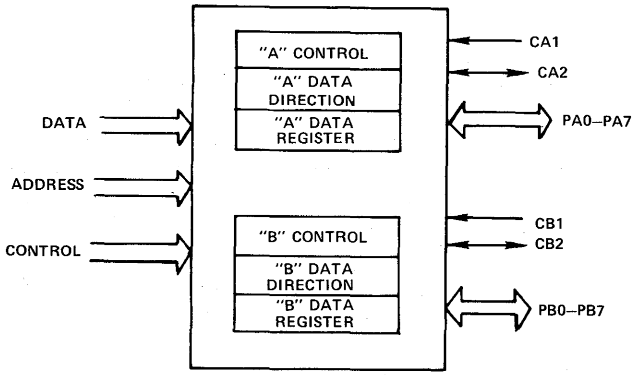


Figure 4—PIA registers

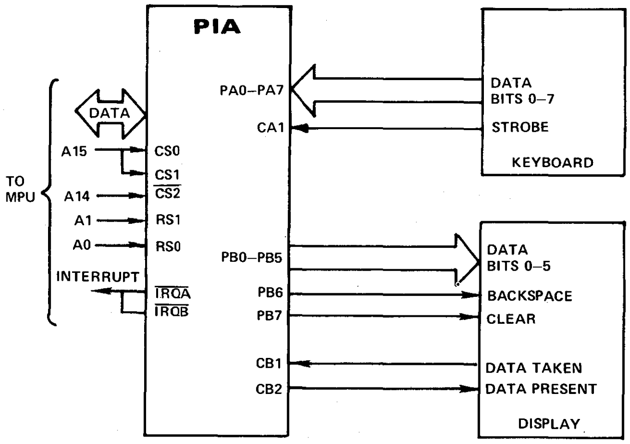


Figure 6—Interfacing to a keyboard and display

AN EXAMPLE OF I/O TRANSFER

Operation of an I/O transfer is best demonstrated by example. Consider the system seen in Figure 6 showing the Motorola PIA interfaced to a keyboard and display. The "A" side of the PIA is tied to the keyboard representing inputs to the MPU. Conversely, the "B" side will be used to output data to a 16 character display. The keyboard interface is comprised of (1) eight data lines tied directly to PIA lines PA0-PA7, and (2) a keyboard strobe to CA1 to inform the MPU of new input data. The display interface uses PB0-PB7 to transfer both data and control functions (backspace and clear). The control lines are configured into a handshake mode with CB2 generating a "data present" signal when new data is displayed. CB1 then acknowledges with "data taken" to complete the handshake loop.

The MPU/PIA interface of Figure 6 shows data, address, and interrupt signals. An important point to mention is that the PIA is assigned a memory area by virtue of the address lines into the device. The chip select inputs (CS0, CS1, and CS2) are used to enable the PIA and the register select inputs (RS0, RS1) specify registers within the PIA. The MPU may, therefore, access the PIA registers with the following addresses:

Address (HEX)	PIA Register	
8000	"A" data direction	("A" control register bit 2=0)
	"A" data	("A" control register bit 2=1)
8001	"A" Control	
8002	"B" data direction	("B" control register bit 2=0)
	"B" data	("B" control register bit 2=1)
8003	"B" control	

Having established the general scenario, the discussion continues with programming examples of the PIA initialization, keyboard input and display output using the Motorola M6800 instruction set.

Initialization

As mentioned previously, the PIA is entirely programmable. The interface seen in Figure 6 requires the following definition of PIA signals:

(1) PA0-PA7 are inputs.
(2) The PIA is to generate an interrupt to the MPU on the rising edge (0→1) of the CA1 input.
(3) CA2 is unused.
(4) PB0-PB7 are outputs.
(5) The PIA is to generate an interrupt to the MPU on a falling edge (1→0) of the CB1 input.
(6) When data is written into PIA (from MPU) to the display, a CB2 strobe occurs.

Programming the PIA interface becomes a series of LOAD and STORE instructions. LOAD brings the desired data pattern into the MPU and STORE ships the word to the addressed PIA registers. The initialization program of the PIA then becomes the following:

```
      Program*     Comments
COM   $8002**      Complement location 8002—defines
                   PB0-PB7 as outputs
LDAA  #$C7***      Load accumulator A with a value of C7
STAA  $8001        Store accumulator A in location 8001—
                   defines CA1 and CA2 characteristics
LDAA  #$ED         Load accumulator B with a value of ED
STAA    $8003      Store accumulator B in location 8003—
                   defines CB1 and CB2 characteristics
```

* All registers are cleared at power-on reset.

** $ denotes hexadecimal.

*** # denotes immediate addressing in which the subsequent character is data instead of address.

Input data transfer

Having initialized the PIA, data is moved from the peripheral to the memory by a sequence of LOAD and STORE instructions. But, the movement of data is only half the story, control of the transfer must also be considered. In the example of Figure 6, the system is assumed to be under interrupt control. When an interrupt occurs the MPU jumps to a defined area of memory to begin the interrupt service program. The service program must first determine the origin of the interrupt by polling the PIAs. The PIA has bit positions reserved in the control registers (CRA7, CRA6, CRB7, CRB6) that the MPU may read to ascertain the source of the interrupt. In our example, if the keyboard "strobes" the PIA, an interrupt is sent to the MPU and bit 7 is set high in the "A" control register (CRA7). Likewise, a "data taken" interrupt from the display will set bit 7 of the "B" control register (CRB7). The "polling" routine therefore becomes:

```
LDAA     $8001  Load contents of "A" control register.
BMI  KEYBRD     If CRA7=1, go to keyboard routine.
LDAA     $8003  Load contents of "B" control register.
BMI  DISPLY     If CRB7=1, go to display routine.
```

The "polling" sequence of the interrupt service program is nothing more than a read of the control register followed by a conditional branch. If the interrupt bits are set the program branches to the appropriate peripheral routine. If the condition is not met, the "polling" routine continues inspection of PIAs. Should additional peripherals be added to the system the software adjusts simply by continuing the poll. Also note the interrupt priority is built into the software by the polling sequence. The order in which the peripherals are polled implicitly defines the priority.

As another alternative, software can poll all peripherals, evaluate the peripherals requesting service, then branch to the desired routine. In this manner the MPU acquires full visibility of outstanding service requests. Based upon the combination of requests the program can then decide which service routine to enter. The important point to remember is the flexibility of I/O transfer under software control.

Having determined the source of the interrupt, the program branches to the desired peripheral routine. If, for example, the keyboard service routine is entered the MPU performs an input data transfer. A single data transfer to memory may be accomplished in two instructions:

```
LDAA   $8000   Load keyboard data.
STAA   $0100   Save in memory location 0100.
```

When the MPU reads the PIA data, the PIA will automatically reset the corresponding control bits (i.e., bits CRA7 and CRA6 are cleared when reading "A" data), and clear the interrupt to the MPU. The interrupt service routine is then completed with a return from interrupt (RTI) instruction. The cycle is complete: the interrupt was acknowledged, PIAs polled for service, selected peripheral serviced, interrupt conditions reset, and the MPU returned to its operating program.

Output data transfer

Data may be transferred from memory to I/O under control similar to data input. Transfer may be initiated by the software or hardware. In this case, the program initiates the data transfer and the display responds with a "data taken" pulse to signify when another character may be sent. The actual data transfer can be done in two instructions:

```
LDAA   $0100   Load data from memory location 0100.
STAA   $8002   Store data into location 8002.
```

The STAA instruction will load new data into the PIA "B" data register. The PIA will then transfer the data and a "data present" pulse to the display.

Block transfer

The I/O transfer demonstrated in the example shows only a single data transfer to give the reader a feel for data movement under program control. To transfer multiple words or large blocks of data between the peripheral and memory, the program requires more management, but the technique remains the same. The programming ease and speed of I/O data transfer becomes largely a function of the MPU under use. MPU features such as available addressing modes and instruction set become important tools for efficient transfer.

As an example of multiple data transfer, the Motorola MC6800 coupled with the PIA can perform an input operation under full program control as shown below:

```
                                                      Time

RDLOOP  LDAA $8001        Read PIA control           4μsec
                          Register
        BPL RDLOOP        Branch to RDLOOP           4μsec
                          if Bit 7 is plus (0)
        LDAA $8000        Read PIA data              4μsec
        STAA OFFSET,X     Store data at address      6μsec
                          defined by [index
                          reg+offset]
        DEX               Decrement index            4μsec
                          register
        BNE RDLOOP        Branch to RDLOOP           4μsec
                          if not equal to zero
                                         Total      26μsec
```

The first two instructions loop until a request for transfer is received from the peripheral. The request is made to the control input, CAl, of the PIA which then sets CRA7 of the control registers. The LDAA/BPL instructions monitor the control register inspecting the most significant bit (CRA7). The program remains in this two instruction RDLOOP until CRA7 is set to a logic 1. When set, the program breaks out of the loop to begin the data transfer. The MPU reads the data from the peripheral at PIA address 8000 hex. When the data is read into the MPU, the PIA automatically resets the CRA7 bit to zero. The data is then stored into memory using the MC6800 indexed addressing mode. Here the OFFSET (byte 2 of the instruction) is added to a 16-bit index register internal to the MPU. The resulting 16-bit word is used to address the destination of the data. The next instruction, DEX, decrements the index register in preparation for the next I/O transfer. The BNE instruction conditionally branches back to the RDLOOP until the index register is decremented to zero. Once back in the RDLOOP the program cycles waiting for another peripheral service request. When the request is made, CRA7 is set and the program enters the next byte into memory. The cycle is continued until the result of the DEX instrumentation is zero. When the index register is decremented to zero the transfer is complete and the routine is exited.

This multiple word transfer through the MPU is a good contrast to the interrupt controlled transfer shown in the keyboard example. Here the interrupts are disabled and the program polls the peripheral for service via the PIA. Notice that the peripheral need not be synchronous with the program due to the RDLOOP. The program can complete the full cycle in 26μsec. If the peripheral is not prepared for another transfer the program will simply cycle in RDLOOP until a peripheral request is made. Also, note that the length of transfer and location of data storage are varied by presetting the index register prior to entry into the routine. The resulting I/O transfer can move a block of data from a peripheral into contiguous memory locations at up to a 38.5K byte rate.

SUMMARY

In microprocessor based systems today, data movement is commonly an important aspect of system operation. As new applications evolve, users will be evaluating microprocessors with a critical eye toward I/O transfer. The number of instructions is less important than the nature of the instruction and usable addressing modes. How quick can the MPU respond to a peripheral interrupt and how is the interrupt managed? What is available from the vendor in the way of interface devices? The list of questions extends in proportion to the needs of the user. As semiconductor houses continue to move into the second and third generation microprocessors, the answers will be easier and faster.

Microprocessors at work—Session overview

CHAIRMAN—PAUL M. RUSSO
RCA Laboratories

MICROPROCESSORS—NO LONGER A NOVELTY

Since the 1971 introduction of the first commercial microprocessor by INTEL, almost every major semiconductor manufacturer has introduced or has under development a "microprocessor" type of device. Microprocessors will be available in most of the existing and future high volume technologies, including PMOS, NMOS, CMOS, Bipolar and I^2L. Microprocessor chips range from 2 or 4 bit slices for Bipolar devices, through 4, 8 and even 16 bit MOS microprocessors on single chips.

Considerable debate is still raging regarding the subtle distinctions between calculator chips, microprocessors, and multi-LSI chip minicomputers. Suffice it to say, that whichever of these three classes of devices one is contemplating using in a given application, many of the major design tradeoffs and system advantages (programmability, flexibility, maintenance, and cost) apply equally well. Microprocessors are no longer a novelty, and the list of products that employ these devices is growing longer every day. It has become almost impossible to pick up a trade journal without coming across several new developments relating to microprocessors. For these reasons, it was felt appropriate to organize a session dedicated not to hypothetical applications and paper designs, but to real world systems that are currently being implemented. Several articles[1,2,3] have recently explored the vigorously developing microprocessor applications areas.

OVERVIEW OF FORMAL PAPERS

The first paper of this session, entitled "The Synergistic Combination of an Oscilloscope and a Microprocessor," by Walter A. Fischer of the Hewlett-Packard Company, explores the use of microprocessors in instrumentation applications. The HP1722A oscilloscope is not the first commercial instrument to utilize a microprocessor, but it represents a major advance to an instrument that has traditionally been the engineer's right hand, and whose basic operation has not changed in many years. As such it typifies what will undoubtedly be a new instrumentation design philosophy.

The second paper of the session entitled "Development of a Portable Computer for Industrial Microcomputer Systems," by Dr. Leroy H. Anderson of the Warner and Swasey Company, covers potential applications of microprocessors in numerical and process control, and defines a unique English-like process control language (PCL), along with a portable PCL compiler highly suited to process control applications. The development of the PCL language may point to a novel approach to microprocessor software development in which the use of high level languages tailored to specific applications will greatly simplify the development of specific system designs.

The sessions' third paper, "Microprocessors in CRT Terminals," by John Whiting and Sandy Newman of Beehive Medical Electronics, covers the broad area of microprocessor applications to CRT terminals. Tradeoffs regarding both the use and choice of microprocessors are discussed. An excellent perspective of what microprocessors can and cannot do in a CRT environment is presented. The trials and tribulations associated with program development and debugging are discussed openly and candidly, and several useful debugging tools are detailed.

The final formal paper of the session, entitled "Designing an Application Oriented Terminal," by J. P. Kohli of the NCR Corporation,* describes the Honeywell 7340 bank teller terminal. The 7340 is a microprocessor-based application oriented terminal for the banking industry, and as such, illustrates a typical terminal application where local intelligence facilitates the processing of transactions. Many of the decisions relating to real time processing, customer programming and system architecture are succinctly described.

MICROPROCESSOR-BASED DATA COMMUNICATIONS SYSTEMS

The one major microprocessor application area that has not been adequately covered by the four formal presentations, is that of data communications. Considerable development work is ongoing in the use of microprocessors in narrowband store-and-forward communications systems[4,5] in intelligent repeaters associated with digital communications links, and in various switching and monitoring applications where the power and cost of minicomputers is not warranted.

Prior to the panel discussion, Dale Walls, of Collins Radio, will present a brief overview of this burgeoning area of microprocessor applications.

* Mr. Kohli was with Honeywell Information Systems, Inc. at the time this paper was written.

RECENT DEVELOPMENTS

The trade journals abound with examples of recent microprocessor-based systems. From assembly line torque monitoring in Detroit (Intel 8080), to the intelligent oscilloscope (HP1722A), to traffic light control (Intel 8008), to a large number of microprocessor-based data terminals (T1742, HP2640A, Beehive Medical Electronics, etc.), and finally to arcade/restaurant TV games, microprocessors are increasingly becoming a part of our daily lives.

The next step beyond a microprocessor-based system is one employing several processors. Several such products already exist. Financial Data Sciences, Inc. Model 108 teller terminal consists of three MCS-4 cpu's. One cpu controls the printer, another controls the keyboard and performs all the required calculations, and another provides stand-alone processing should the communications link to the main cpu fail. Another example is the OP-1 CRT terminal from Ontel. It sports an Intel 8008 as the central processor and uses two other (TTL MSI) processors to control the I/O and keyboard/display operations. These multi-microprocessor systems are but a preview of what will surely come.

FUTURE APPLICATIONS

As microprocessors evolve, and price/performance improves, many new applications areas will emerge. The upper end of the microprocessor performance spectrum will be used to implement many systems currently employing minis, and may also be used in the development of programmable high performance peripheral interfaces. The use of medium range microprocessors will accelerate in the various industrial applications areas typified by the applications presented in this session. Potentially, the real dollar growth in applications will occur in the consumer[6] and automotive areas,[7] where the lower end of the microprocessor performance spectrum should prove more than adequate to satisfy the requirements of the bulk of the systems envisioned.

The consumer and automotive computer markets are extremely price sensitive and do not require excessively high performance. Thus, in order to be successful in this area it is incumbent on the semiconductor manufacturers to introduce new microprocessor products which achieve a given performance level, but do so at minimum system cost. Price performance must be improved, but this should be achieved by lowering cost for a given performance level, rather than by increasing performance for a given price level!

Finally, before closing, the following observation should be made. Even though the microprocessor is the key to the development of many new low cost intelligent systems, it is the development of compatible low cost peripherals (e.g., floppy discs) and LSI memories (static 1K RAM's are approaching 0.4 cents/bit and dynamic 4K RAM's are approaching 0.3 cents/bit at the chip level) that is making this system revolution possible. Recent developments such as the modem-on-a-chip (Motorola MC6860), point to the availability of a large variety of standard LSI microprocessor interface, which can only help to accelerate this revolution.

REFERENCES

1. "Special Report: Microprocessor Applications," *Electronics,* Vol. 47, No. 14, July 11, 1974, pp. 81-108.
2. "Special Issue on Microprocessor Applications," *IEEE Computer,* Vol. 7, No. 8, August 1974, pp. 19-53.
3. "Microprocessor Applications," *Spectrum,* September 1974, pp. 59-67.
4. Russo, P. M. and M. D. Lippman, "A Microprocessor Implementation of a Dedicated Store-and-Forward Data Communications System," *AFIPS Conference Proceedings,* Volume 43, May 1974.
5. Russo, P. M. and M. D. Lippman, "Case History: Store and Forward," *IEEE Spectrum,* September 1974, pp. 60-67.
6. Weisbecker, J. A., "A Practical, Low Cost, Home/School Microprocessor System," *IEEE Computer,* August 1974, pp. 20-31.
7. Temple, R. H. and S. S. Devlin, "The Use of Microprocessors as Automobile On-Board Controllers," *IEEE Computer,* August 1974.

The synergistic combination of an oscilloscope and a microprocessor

by WALTER A. FISCHER
Hewlett-Packard Company
Colorado Springs, Colorado

INTRODUCTION

Oscilloscopes—What they are and what they do

An oscilloscope presents a graphical display of amplitude vs. time. The amplitude is usually voltage. It allows the electrical designer to see what is occurring in a circuit. The CRT display was originally a qualitative one and provided the designer with an idea of what was occurring. Through improvements in vertical amplifier design, sweep linearity, and CRT performance, a calibrated graticule was added to the CRT face and quantitative measurements could be made. These improvements continued and resulted in measurement accuracies in the 2 percent to 3 percent category, with some timing measurements reaching the 1 percent area. These accuracies represent the state-of-the-art performance with traditional techniques. A new set of techniques was becoming an obvious need in order to make major improvements in the measurement capabilities of oscilloscopes.

Where major improvements in measurements are needed

The two main categories where improvements are needed are measurement accuracy and ease of use.

Measurement accuracy

Oscilloscopes measure voltage and time related functions; such as, peak-to-peak voltage, percent overshoot, periods, propagation delay, etc. Timing measurement accuracy is the area where most customers have requested improvements. Specifically, the area of propagation delay. The reason is that a major part of electrical design tasks today are oriented to digital designs. One of the most important requirements for proper digital circuit performance is that information arrive at the various nodes in the system at a precise time. Even if the amplitude is in error, or contains overshoot, as long as the signal is timed properly at the logic threshold level, a good signal will be recorded. If, the threshold level arrives at the wrong time, this can cause major failures in system performance. It is necessary to measure precisely, the relative time delays of signals arriving at a point through different paths. Pulse width, period, transition times, and clock rate must also be measured.

The oscilloscope is still the best form of instrumentation to measure instantaneous voltages. It is also used to measure dc voltages as well as percent overshoots, and logic threshold levels. These measurements can now be made with oscilloscopes but not with any amount of ease and are subject to considerable human error. Such things as counting graticule lines and multiplying by the sensitivity of the CRT take time and are subject to human error.

Ease of use

One of the features of an oscilloscope is its versatility in making a large variety of measurements. This versatility has always required a large number of front-panel controls. This is its biggest problem. Most of these controls are manual, not only in function but also in their ability to allow the operator to make measurements, therefore it requires a great deal of thought on the part of the operator just to use the scope. It is possible on most oscilloscopes through a combination of controls to achieve completely useless modes. Even more of a problem is the fact that gross measurement errors can occur when the oscilloscope is in any of the "uncal" modes.

These are just a few areas where major improvements in ease of use can be made.

A SOLUTION TO THE MEASUREMENT ACCURACY AND EASE-OF-USE PROBLEM

The newly introduced HP Model 1722A is a synergistic combination of an oscilloscope and a microprocessor and makes major contributions in measurement accuracy and ease of use. It is basically a 275 MHz high-performance oscilloscope with up to 1 nanosecond per cm resolution in the time base.

The major contribution is timing measurement accuracy. Two things contribute to this, they are dual-delay sweep* and microprocessor control.

Dual-delay sweep is a technique that allows the operator

* Patent applied for.

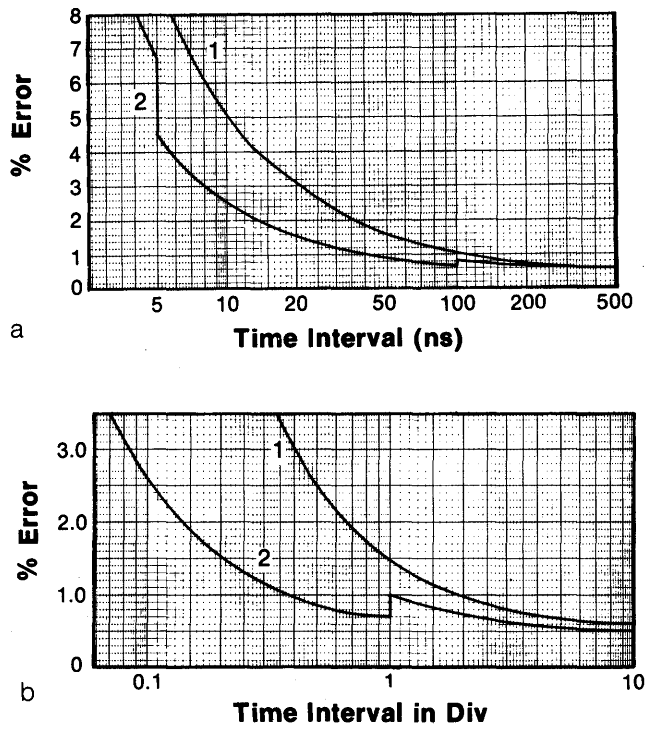


Figure 1—A comparison of the accuracy of time interval measurements of the HP Model 1722A and a conventional oscilloscope (a) Error curves for time intervals from 1 ns to 500 ns for (1) Conventional scope using differential delay techniques; (2) 1722A specification. Curves derived from optimum main time base settings for this measurement range. (b) Error curves for time intervals in terms of main time base divisions (100 ns/div to 20 ms/div)

to see, simultaneously, both the start and stop points of the time interval being measured whether it be period, pulse width, propagation delay, etc. This automatically eliminates the CRT as well as vertical or horizontal amplifier drift (or both) as sources of error.

The microprocessor adds an order of magnitude (more accuracy) to the standard oscilloscope by providing greater resolution and readability than had previously been possible. Specifically, better than 1 percent measurements can be made on time intervals as small as 30 nanoseconds or 4 percent of full scale. Figure 1 shows a comparison of accuracy in graphical form of the Model 1722A and a high-quality standard oscilloscope of equivalent bandwidth.

The microprocessor also presents direct digital readout of all measurements. Table I lists the measurement set of the Model 1722A.

The gross measurement errors and useless modes previously referred to are remedied by the microprocessor. The Model 1722A monitors various front-panel controls and, when necessary, prevents incorrect measurements from being made. For example, when making a timing measurement, if the sweep is set to the "uncal" mode, the microprocessor senses this and sets the LED readout to (.0) and eliminates the stop marker of the dual-delayed sweep markers. In the vertical section, when the vernier is placed in the "uncal" mode, the instrument automatically goes into the percentage measuring mode.

These are just a few of the advantages of using a microprocessor in instrumentation.

WHY A MICROPROCESSOR INSTEAD OF COMBINATIONAL LOGIC?

There is no clear-cut choice. There are some advantages and disadvantages to each of these approaches. Combinational logic, because it is traditional, is often chosen when another approach should be considered. When many functions are required a large number of components are

TABLE I—HP Model 1722A Measurement Set

I. Time Interval

A. Period

B. Transition times

C. Propagation delay

II. 1/Time

A. Clock rate

B. Data rate

III. DC volts

A. Average voltage

B. Direct difference voltage

IV. Instantaneous volts

A. Peak-to-peak

B. Threshold voltage

V. Percent readout

A. Percent overshoot

B. Percent transition times

C. Identifying 50% points on pulses

necessary. This can lead to power and heat problems. The major advantages of combinational logic for small systems are knowledge and availability of components. Most electrical designers feel comfortable with this approach because they have used it traditionally. If only a few functions are required, combinational logic can be the best choice.

The microprocessor approach, however, can make many functions available using fewer components. This usually results in higher reliability and lower power consumption. The major advantage of the microprocessor approach is its ability to perform mathematical operations. Many of the algorithms used by the Model 1722A require the use of mathematics (refer to the section on algorithms). Its major drawback is non-familiarity. The average electrical designer has little or no experience in programming at the assembly level and therefore tends to avoid it. In the past, it has been difficult to justify the training costs in light of the profit motivation of industry. This situation seems to be improving, however, as it becomes obvious that microprocessor based control systems can be inexpensive, reliable, and add measurement capability never before available.

At this point, a case history might prove interesting.

The curve in Figure 2 gives an indication of the decision to be made. With combinational logic, the cost increases in a somewhat linear fashion, depending on the number and complexity of the functions desired. With the microprocessor, there is a minimum amount of hardware necessary even if only one function is performed. As the number of functions increase, the cost increases at a rate far less than that of combinational logic. The steps indicate when a new block of memory needs to be added. This is because memory cannot be bought one word at a time but must be bought in blocks; e.g., 256 × 8 bits. An interesting area on these curves is the intersection. It is here that the microprocessor approach becomes obviously less costly than the combinational logic approach.

In the case of the HP Model 1722A, this occurred in the display function. One of the requirements for this instrument was that the LED display present answers in a very special form of scientific notation; namely that the exponent could take on only values which were multiples of the number three. The time interval always can be read directly in s (0), ms (−3), μs(−6) or ns (−9). This is shown in Figure 4 and Table II. The cost of doing this with combinational logic was high—therefore, the microprocessor approach was considered and found to reduce cost and package count, with far less power required. The choice of which approach to use must be made carefully. You may be surprised at the small number of functions it takes to justify a microprocessor-based system.

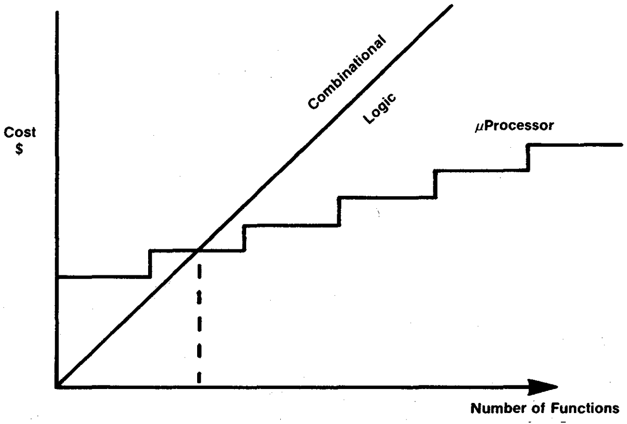


Figure 2—A comparison of the costs involved between combinational logic and μ processor based systems

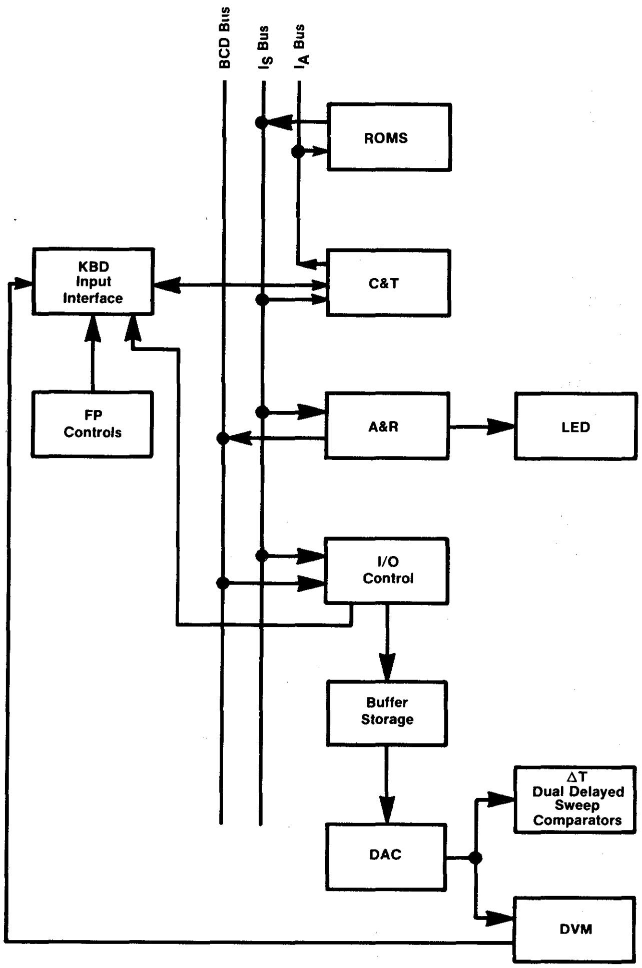


Figure 3—Block diagram of the HP Model 1722A μ processor based control system

BLOCK DIAGRAM OF THE MICROPROCESSOR-BASED SYSTEM USED IN THE HP MODEL 1722A

The technique employed in the Model 1722A was to use the microprocessor LSI circuits from the HP-35 calculator with a unique set of ROM's programmed to perform the

TABLE II—Time Base Encoder Program Listing

ROM Address	ROM Code	ROM Subroutine Address	Labels	Program Statement
L1006:	. . . 1 . 1 1 . . .		LSCA4	: LOAD CONSTANT 1
L1007:	. . 1 . . . 1 . 1 1	->L1042		GO TO LS03
L1010:	. . 1 . . 1 1 . . .		LSCA5	: LOAD CONSTANT 2
L1011:	. . 1 . . . 1 . 1 1	->L1042		GO TO LS03
L1012:	. . 1 . . . . 1 1 1	->L1041	LSCA6	: GO TO LS02
L1013:	. . . 1 . 1 1 . . .		LSCA7	: LOAD CONSTANT 1
L1014:	. . 1 . . 1 1 . . .		LSCA8	: LOAD CONSTANT 2
L1015:	. . 1 . 1 . . . 1 1	->L1050		GO TO LS05
L1016:	. 1 . 1 . 1 1 . . .		LSCA9	: LOAD CONSTANT 5
L1017:	. . 1 . 1 . . . 1 1	->L1050		GO TO LS05
L1020:	. . . 1 . 1 1 . . .		LSCB0	: LOAD CONSTANT 1
L1021:	. . 1 . . 1 1 . 1 1	->L1046		GO TO LS08
L1022:	. . 1 . . 1 . . 1 1	->L1044	LSCB1	: GO TO LS06
L1023:	. . 1 . . 1 . 1 1 1	->L1045	LSCB2	: GO TO LS07
L1024:	1 . . 1 . 1 1 . . .		LRNG1	: LOAD CONSTANT 9
L1025:	. . . . 1 1	->L1000		GO TO LRTN0
L1026:	. 1 1 . . 1 1 . . .		LRNG2	: LOAD CONSTANT 6
L1027:	. . . . 1 1	->L1000		GO TO LRTN0
L1030:	. . 1 1 . 1 1 . . .		LRNG3	: LOAD CONSTANT 3
L1031:	. . . . 1 1	->L1000		GO TO LRTN0
L1032:	. . . . 1 1	->L1000	LRNG4	: GO TO LRTN0
L1033:	1 . . . . . 1 1 . .		LKBD1	: 8->P
L1034:	. . . .			NO OPERATION
L1035:	. . 1 1 . 1			KEYS->ROM ADDRESS
L1041:	. 1 . 1 . 1 1 . . .		LS02	: LOAD CONSTANT 5
L1042:	. . 1 1 . . 1 . . 1	->L1062	LS03	: JSB LDP4
L1043:	. . 1 . 1 . . 1 1 1	->L1051	LS04	: GO TO LKBD2
L1044:	. . 1 . . 1 1 . . .		LS06	: LOAD CONSTANT 2
L1045:	. 1 . 1 . 1 1 . . .		LS07	: LOAD CONSTANT 5
L1046:	. . 1 . 1 1 1 . . 1	->L1056	LS08	: JSB LDP2
L1047:	. . 1 . 1 . . 1 1 1	->L1051		GO TO LKBD2
L1050:	. . 1 1 1	->L1060	LS05	: JSB LDP3
L1051:	. . . 1 . . 1 1 . .		LKBD2	: 1->P
L1052:	1			ROM ADDRESS->BUFFER
L1053:	. . 1 1 . 1			KEYS->ROM ADDRESS
L1056:	1 . 1 . . . 1 1 . .		LDP2	: 10->P
L1057:	. . 1 1 . . 1 1 1 1	->L1063		GO TO LDP0
L1060:	1 . 1 1 . . 1 1 . .		LDP3	: 11->P
L1061:	. . 1 1 . . 1 1 1 1	->L1063		GO TO LDP0
L1062:	1 1 . . . . 1 1 . .		LDP4	: 12->P
L1063:	. . 1 . . 1 1 . . .		LDP0	: LOAD CONSTANT 2
L1064:	. . . . 1 1			RETURN

functions needed to accomplish the measurements listed in Table I. With this in mind let us discuss the block diagram of Figure 3.

The primary function of the processor (Arithmetic & Register and Control & Timing) is to continuously scan the appropriate front-panel controls and output the proper signals to both the LED display and to the oscilloscope circuits. The front-panel controls, therefore, are essentially a keyboard similar to the keyboard of the HP-35 and as such their outputs are encoded by the input interface to present particular memory addresses to C & T. Programs are stored at these addresses and perform the appropriate functions, such as, increment, decrement, output to the display, compute a time, etc.

The BCD output of A & R is directed to the I/O control where two things occur. First, if data are being output, it converts the data from serial to parallel data and transfers them to buffer storage. Second, if the front panel controls are to be scanned, it decodes the outputs from the Processor and enables the appropriate sections of the front panel; such as, vertical range, timebase range, etc.

The Buffer Storage and DAC receive data from the I/O control and provide temporary data storage and conversion to analog levels for the Analog Amplifier assembly.

The Analog Amplifier performs two functions. First, it supplies the dual-delayed sweep comparators with the proper dc levels. Second, it accepts the dc level from the vertical channel, processes this level and provides two pieces of information for the processor through the Input Interface. The two pieces are the polarity of the dc level and whether the level is greater or less than some reference. If it is greater, the processor increases the reference until it is within 1LSB of the unknown. Conversely, if it is less, the processor decreases the reference until it is within 1LSB of the unknown. In both cases, it displays the reference level that is now equal to the unknown.

SERIAL MICROPROCESSOR FOR OSCILLOSCOPE USE

There are many microprocessors available on the market; why then, choose the serial microprocessor? One reason was the fact that the HP-35 microprocessor was available as a high volume, fully documented microprocessor. Below are additional reasons for this choice.

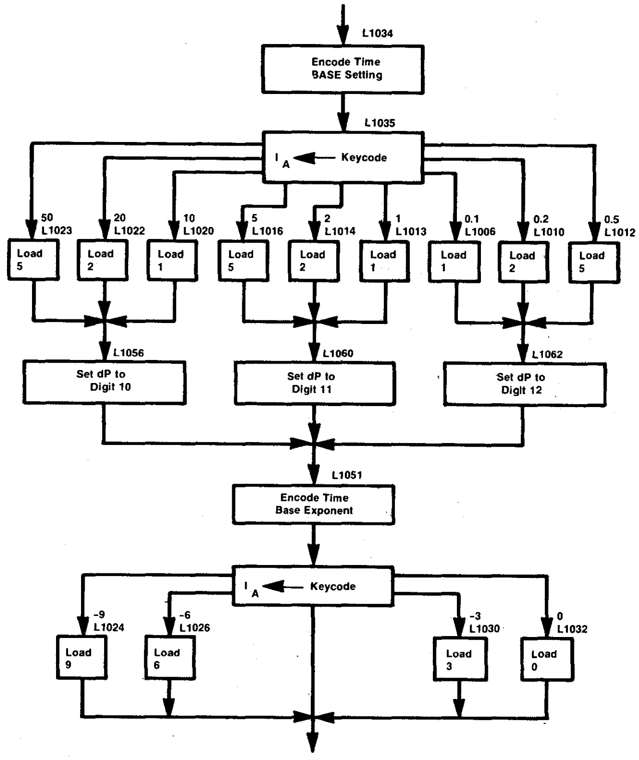


Figure 4—Time base encoder flow diagram

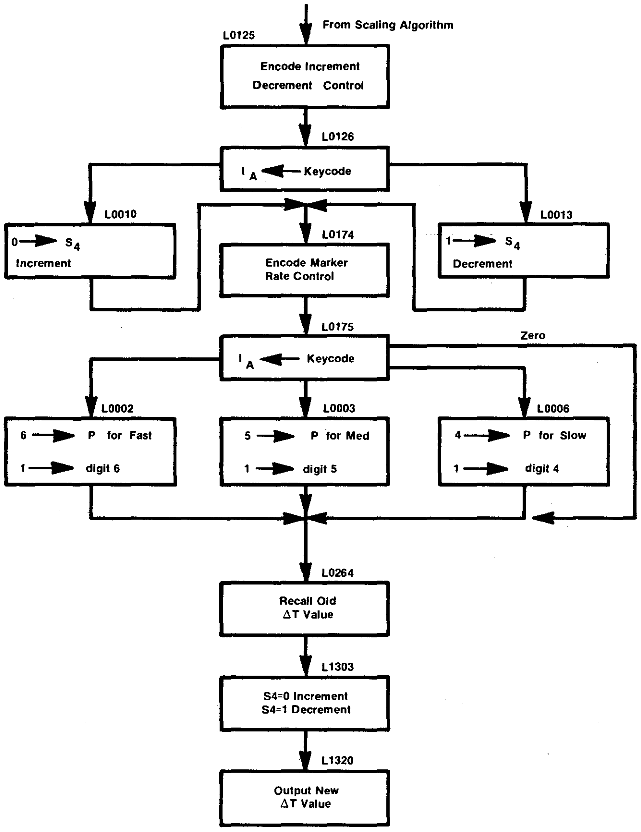


Figure 5—Data output algorithm for Δ T mode of 1722A

Display functions

This was one of the major cost justifications for using a microprocessor. In the HP-35 serial microprocessor the complete decoder system, compatible with the basic instruction set, is resident in the LSI arithmetic and register circuit. A set of bi-polar cathode/anode drivers are available, and sign and decimal location are also part of this chip set.

Keyboard scanning

The keyboard scanning circuits are resident in the LSI and with one keyboard enable, 40 keycode inputs are possible. Since some of the internal status bits are available to the software programs, several keyboards can be overlayed with this approach. In the HP Model 1722A, for example, there are a total of six keyboards. The conversion of keycodes to ROM address is done within the control and timing circuit. Figures 4 and 5 and Tables II and III show that the program branches on an externally generated address; e.g., in Figure 4 at L1035.

TABLE III—Program Listing of Data Output Algorithm for 1722A

ROM Address	ROM Code	ROM Subroutine Address	Labels	Program Statement
L0002:	. 1 1 1 1 1 1 . 1 1	->L0176	LFST1	: GO TO LFSTM
L0003:	1 . . . . . . 1 1 1	-> L0201	LMED1	: GO TO LMEDM
L0004:	1 . . 1 . . . 1 . .		LZRO1	: 1 -> S9
L0005:	1 . . . 1 . . 1 1 1	-> L0211		GO TO LMODZ
L0006:	. 1 . . . . 1 1 . .		LSL01	: 4 -> P
L0007:	1 . . . . 1 . . 1 1	-> L0204		GO TO LSL02
L0010:	. . 1 1 . . . 1 . .		LINCP	: 1 -> S3
L0011:	. 1 . 1 1 . . . 1 1	-> L130		GO TO LINC1
L0013:	. 1 . 1 1 . 1 . 1 1	-> L0132	LDECP	: GO TO LINC3
L0024:	. 1 . 1 . 1 . 1 . 1	-> L0125	LOK	: JSB LKBD4
L0025:	. 1 1 1 1 1 . . 1 1	-> L0174		GO TO LKBD6
L0125:	1 1		LKBD4	: ENCODE INCR/DECR CONTR.
L0126:	. . 1 1 . 1			IA KEYCODE
L0130:	. 1 . . 1 . . 1 . .		LINC1	: 0 -> S4
L0131:	. . . . 1 1			RETURN
L0132:	. . 1 1 . . . 1 . .		LINC3	: 1 -> S3
L0133:	. 1 . . . . . 1 . .		LINC4	: 1 -> S4
L0134:	. . . . 1 1			RETURN
L0160:	. . 1 1 . . 1 . 1 .		LAEC	: 0 -> C[X]
L0161:	. 1 1 1 . . 1 . 1 .			A + C -> C[X]
L0162:	1 1 1 . 1 1 . . 1 .			A EXCHANGE C[WP]
L0163:	. . . . 1 1			RETURN
L0174:	1 . 1		LKBD6	: ENCODE MARKER RATE
L0175:	. . 1 1 . 1			IA KEYCODE
L0176:	1 . . 1 . . . 1 . .		LFSTM	: 1 -> S9
L0177:	. 1 1 . . . 1 1 . .		LFSTN	: 6 -> P
L0200:	1 . . . 1 . . . 1 1	-> L0210		GO TO LSL03
L0201:	1 . . 1 . . . 1 . .		LMEDM	: 1 -> S9
L0202:	. 1 . 1 . . 1 1 . .		LMED2	: 5 -> P
L0232:	1 . . . 1 . . . 1 1	-> L0210		GO TO LSL03
L0204:	1 . . 1 . 1 . 1 . .		LSL02	: IF S9 # 1
L0205:	1 . . . 1 . . . 1 1	-> L0210		THEN GO TO LSL03
L0206:	1 . 1 . . . . 1 . .			1 -> S10
L0207:	1 . . 1 1 . . 1 . .			0 -> S9
L0210:	. . . 1 . 1 1 . . .		LSL03:	: LOAD CONSTANT 1
L0211:	. 1 1 1 . . 1 1 . .		LMODZ	: 7 -> P
L0212:	. . . 1 . 1 . 1 . .			IF S1 # 1
L0213:	1 . 1 1 . . 1 1 1 1	-> L0263		THEN GO TO LMOD0
L0263:	. 1 1 1 1	-> L0160	LMOD0	: JSB LAEC
L0264:	1 1 . . 1 . 1 . . .			DOWN ROTATE
L0265:	. 1 1 1 1	-> L0160		JSB LAEC
L0266:	1 1 1 1 1 . 1 . 1 1	-> L0372		GO TO LMODA
L0372:	. . 1 . . 1	-> L1373	LMODA	: SELECT ROM 1
L1303:	. . 1 1 . . 1 . 1 .		LMOD1	: 0 ->8 C[X]
L1304:	. 1 . . . 1 . 1 . .			IF S4 # 1
L1305:	1 1 . . 1 . 1 1 1 1	-> L1313		THEN GO TO LMOD2
L1306:	. 1 . 1 . 1 . . 1 .		LMOD4	: A — C -> C[WP]
L1307:	1 1 . . 1 1 1 1 1 1	-> L1317		IF NO CARRY GO TO LTB3
L1310:	. . 1 1 . 1 . . 1 .		LMIN	: 0 -> C[WP]
L1311:	1 1 . . 1 1 1 1 1 1	-> L1317		IF NO CARRY GO TO LTB3
L1312:	1 1 . . 1 1 1 1 1 1	-> L1317		GO TO LTB3
L1313:	. 1 1 1 . 1 . . 1 .		LMOD2	: A + C -> C[WP]
L1314:	1 1 . . 1 1 1 1 1 1	-> L1317		IF NO CARRY GO TO LTB3
L1315:	. . 1 1 . 1 . . 1 .		LMAX	: 0 -> C[WP]
L1316:	. . 1 1 1 1 . . 1 .			0 — C — 1 -> C[WP]
L1317:	1 . . . . . 1 1 . .		LTB3	: 8 -> P
L1320:	1 1 1			DATA OUTPUT
L1373:	1 1 . . . . 1 1 1 1	-> L1303		GO TO LMOD1

The connection requirements are minimal with this chip set since only 13 lines are required for all 40 keycode inputs.

Large word length

This is one of the key features to the serial microprocessor. The word length is 56 bits (14 digits) and one instruction can work on the whole word or a variety of parts of the word; e.g., exponent only. Therefore, a large number of control functions as well as data can be output to the oscilloscope in only one word time, thus resulting in an efficient transfer of control and data.

BCD arithmetic

This advantage may not be obvious immediately. However, in an application where decimal information is the desired output, it means that software manipulation is very efficient because no code conversions are necessary. In the Model 1722A, the DAC, described in the block diagram discussion, is a BCD DAC. Therefore when the manipulation of data is complete, it can be transferred directly. The increment/decrement algorithm demonstrates this in Figure 5 and Table III. Here, the appropriate digit is incremented by 1 and outputted directly with no further code conversions necessary.

Serial interface

In any system that has limitations on space and weight, as in instrumentation, any reduction in parts count and/or cabling is a significant advantage. The serial microprocessor provides an interface that requires few bus lines with the ability to provide simple remote storage with shift registers. The serial I/O also reduces the hardware requirement.

No RAM required

The serial nature of the chip set allowed shift registers to be designed into the arithmetic and register circuit. These are used to store intermediate calculations. In other microprocessors, RAM is required for this function. Again, the volume and number of interconnections are minimized with use of this microprocessor.

Large ROM space available

This is an important feature. In an oscilloscope, there are many controls on the front panel as well as many possible measurement modes as we discussed earlier. Since many of them are interrelated, using one front-panel control may have implications to others. The availability of large ROM space allows programs to be written that take these interrelationships into account.

Instruction set

This is probably the most important consideration. There is no advantage in having a higher-speed parallel microprocessor as a controller if the basic instruction set is limited, and only minor manipulation of data can be performed. This implies that many word-times would be required to perform the more complicated functions.

Even though the HP-35 chip set is serial and has a word-time of 280 μs, the instruction set is so powerful that one instruction can change the entire nature of the next word. In many cases, less time is required to perform basic functions with the HP-35 microprocessor than with competitive parallel processors. For example, a six-digit add requires one ROM state and takes 280 μs. Other price competitive microprocessors require anywhere from 5 to 20 ROM states and can take as long as 800 μs.

This instruction set includes a very complete group of branching instructions which allows subroutines to be easily written.

Software and editing

As in any processor-based system, the need to write and edit software easily is important. The HP-35 microprocessor compilers are written in such a way that single step and dynamic debugging are possible.

ALGORITHMS USED IN THE MODEL 1722A

This section describes in considerable detail three of the algorithms used in the Model 1722A. These three were picked to demonstrate the power of the instruction set, the mathematics, the time base encoding scheme and the efficiency of the data transfer algorithm. They also demonstrate the overall efficiency of the use of ROM states.

Time base encoding

The requirement here is to encode nine time base settings and four exponent values into ROM addresses. This is easily accomplished since the keyboard scanning technique implemented in the C & T chip accepts a keycode entry and uses it as the next address on the IA line (see Figure 3).

Specifically, when the Model 1722A program reaches the point where the time base setting needs to be interrogated, an instruction is generated on IS (Figure 3) that is decoded by the I/O control. The I/O control then enables that part of the keyboard that is monitoring the time base switch setting. The input interface (Figure 3) generates a keycode from which the C & T generates the next address. The detailed algorithm is shown in Figure 4 and Table II.

The important point here is that only 33 ROM states are needed to encode the 9 time base settings and 10 are

TABLE IV—Program Listing of Math and Display Algorithm, for the Time Interval Mode of the 1722A

ROM Address	ROM Code	ROM Subroutine Address	Labels		Program Statement
L0160:	. . 1 1 . . 1 . 1 .		LAEC	:	0 -> C[X]
L0161:	. 1 1 1 . . 1 . 1 .				A + C -> C[X]
L0162:	1 1 1 . 1 1 . . 1 .				A EXCHANGE C[WP]
L0163:	. . . . 1 1				RETURN
L0267:	1 . 1 . . 1 . 1 . .		LSL04	:	IF S10 # 1
L0270:	1 . 1 1 1 . 1 1 1 1	-> L0273			THEN GO TO LSCA0
L0273:	1 . . . . . 1 1 . .		LSCA0	:	8 -> P
L0274:	. . 1 1 . 1 . . 1 .				0 -> C[WP]
L0275	. 1 1 1 1	-> L160	LSCA1	:	JSB LAEC
L0276:	. . 1 1 . . . . 1 .				0 -> C[P]
L0277:	. . 1 1 . . 1 . 1 .		LSCA2	:	0 -> C[X]
L0300:	1 1 1 1 . 1 . . 1 .				A + C -> A[WP]
L0301:	. . 1 . 1 . 1 . . .				C EXCHANGE M
L0302:	. 1 . 1 1 . . . 1 .				C — 1 -> C[P]
L0303:	. 1 1 . 1 . . . 1 .				IF C[P] = 0
L0304:	1 1 . . . 1 1 1 1 1	-> L0307			THEN GO TO LSCA3
L0305:	. . 1 . 1 . 1 . . .				C EXCHANGE M
L0306:	1 . 1 1 1 1 1 1 1 1	-> L0277			GO TO LSCA2
L0307:	. . . 1 . 1 . 1 . .		LSCA3	:	IF S1 # 1
L0310:	1 1 . 1 . 1 . 1 1 1	-> L0325			THEN GO TO LTB4
L0321:	1 . . 1 . . 1 . 1 .		LNEXP	:	SHIFT RIGHT C[X]
L0322:	. . 1 1 . 1 1 . 1 .				0 -> C[XS]
L0323:	. . 1 1 1 1 1 . 1 .				0 — C — 1 -> C[XS]
L0324:	. . . . 1 1				RETURN
L0325:	. . 1 1 . 1 1 1 1 .		LTB4	:	0 -> C[S]
L0326:	. . 1 . . 1 . 1 . .				IF S2 #1
L0327:	1 1 . 1 1 . . 1 1 1	-> L0331			THEN GO TO LTB5
L0330:	1 1 1 1 1 1 . . 1 1	-> L0374			GO TO LINV3
L0331:	1 1 . 1 . . . 1 . 1	-> L0321	LTB5	:	JSB LNEXP
L0332:	1 . . . . . 1 1 . .		LDISP	:	8 -> P
L0333:	1 1 1 . 1 1 1 1 1 .				A EXCHANGE C[S]
L0334:	1 1 1 . 1 . 1 . 1 .				A EXCHANGE C[X]
L0335:	. . 1 1 . 1 1 1 1 .				0 -> C[S]
L0336:	. . 1 1 . 1 . . 1 .				0 -> C[WP]
L0337:	1 . . . 1 . 1 . . .				DISPLAY OFF
L0340:	1 . . . 1 . 1 1 1 .				B EXCHANGE C[W]
L0341:	. 1 . . . . . 1 1 .				SHIFT LEFT A[M]
L0342:	. 1 . . . . . 1 1 .				SHIFT LEFT A[M]
L0343:	. 1 . . . . . 1 1 .				SHIFT LEFT A[M]
L0344:	. 1 . . . . . 1 1 .				SHIFT LEFT A[M]
L0345:	. . . . 1 . 1 . . .		LDSP2	:	DISPLAY TOGGLE
L0373:	1 . 1 1 . 1 1 1 1 1	-> L0267			GO TO LSL04
L1325:	1 1 . . 1 . 1 . . .				DOWN ROTATE
L1326:	1 1 . . 1 . 1 . . .				DOWN ROTATE
L1327:	1 1 . . 1 . 1 . . .				DOWN ROTATE
L1330:	. . . .				NO OPERATION
L1331:	. 1 1 1 . . 1 1 . .				7 -> P
L1332:	. . 1 1 . . 1 . 1 .				0 -> C[X]
L1333:	. 1 1 1 . . 1 . 1 .				A + C -> C[X]
L1334:	1 1 1 . 1 1 . . 1 .				A EXCHANGE C[WP]
L1335:	. 1 1 . . 1 . . 1 .				C -> A[WP]
L1336:	. 1 1 . 1 . . . 1 .		LMOD5	:	IF C[P] = 0
L1337:	1 1 1 . . . . 1 1 1	-> L1341			THEN GO TO LMOD6
L1340:	1 1 1 1 1 . 1 . 1 1	-> L1372			GO TO LSCA0
L1372	. . . . . 1	- > L0373	LSCAO	:	SELECT ROM0

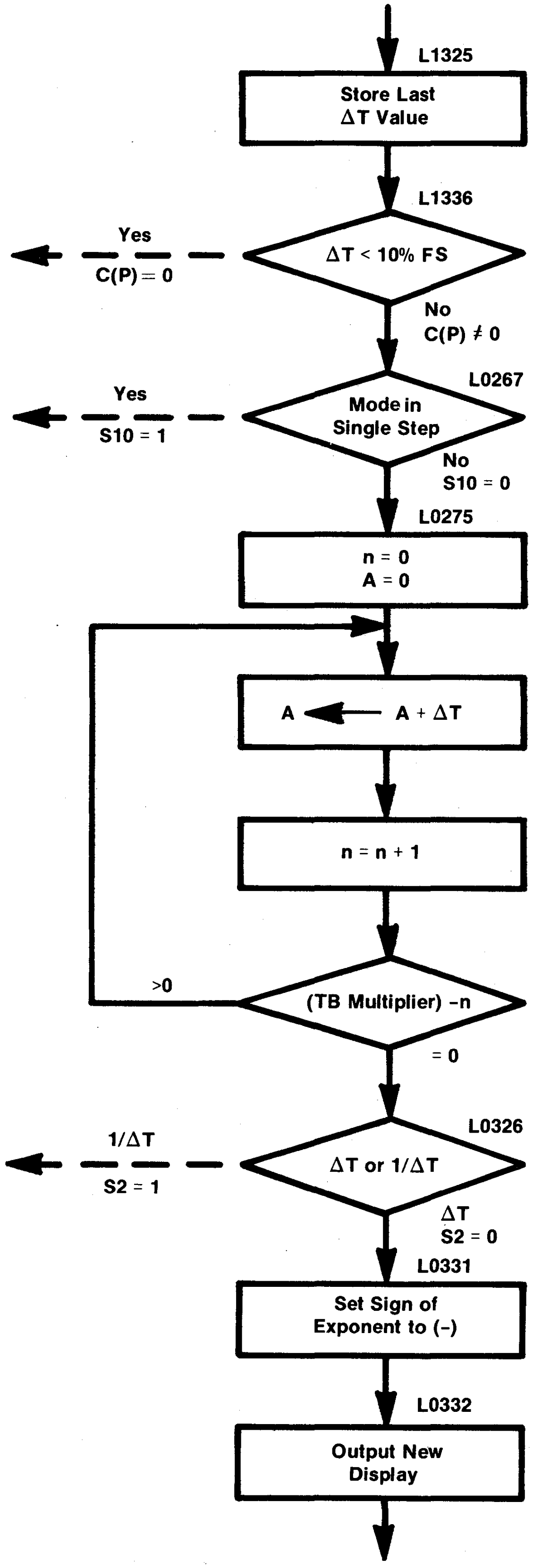


Figure 6—Mathematics and display output algorithm for the time interval mode of the 1722A

needed to encode the time base exponent values. This results in extremely efficient use of ROM states.

These 43 ROM states have encoded and stored the multipliers, the position of the decimal point, and the exponent value, in approximately 5 ms.

Data output algorithm

The requirement here is that the increment/decrement control be encoded, the appropriate corrections be made to the data, and the data outputed to the DAC (see Figure 3).

Two pieces of information need to be encoded. They are (1) should the data be increased or decreased, and (2) at what rate? The first is done starting at address L0125, the second at L0174. Once this is accomplished the appropriate mathematics takes place at address L1306 or L1313 and the new value outputed at L1320. See Figure 5 and Table III.

Thus, the entire encoding, mathematical manipulation, and data output is accomplished with 58 ROM states in approximately 10 ms.

Time internal display algorithm

This algorithm takes the new ΔT value computed in Figure 5 and performs the appropriate mathematical scaling determined from the scaling algorithm in Figure 4. This scaled value is then shifted into the display. See Figure 6 and Table IV.

The most important thing here is that 5-digit multiplication (L0277) takes place with 8 ROM states in less than 12 ms.

OTHER CURRENT MICROPROCESSOR APPLICATIONS IN INSTRUMENTATION

There are many examples of ROM-based control systems in instrumentation today (see references). Some of them, such as, the HP Model 1722A[1] oscilloscope, HP Model 3380A[1] Integrator, and the Tektronix DPO[7], use microprocessor chip-sets found more commonly in hand held calculators, point-of-sale terminals, etc. The remaining instruments, such as, the HP Model 3490A[4] voltmeter and the HP Model 3330A[3] synthesizer, as well as many more, use dedicated, ROM-based microcontrollers designed with off the shelf logic. In either case, the trend toward ROM-based controllers in instrumentation is definite.

CONCLUSION

It is becoming obvious in almost all forms of electrical design that the microprocessor can be an invaluable asset, as it allows "smart" circuits to be developed. The HP Model 1722A is one example of this. The microprocessor will provide the basis for many exciting designs in the future. We at Hewlett-Packard are dedicated to solving customer measurement needs and the microprocessor will play a large role in this.

REFERENCES

1. Fischer, W. and W. Risley, "Improved Accuracy and Convenience in Oscilloscope Timing and Voltage Measurements," *Hewlett-Packard Journal,* December 1974.
2. Whitney, T., France Rode and Chung Tung, "The 'Powerful Pocket-ful': An Electronic Calculator Challenges the Slide Rule," *Hewlett-Packard Journal,* June 1972.
3. Kingsford-Smith, C., "The Incremental Sweep Generator—Point by Point Accuracy with Swept-Frequency Convenience," *Hewlett-Packard Journal,* July 1972.
4. Thompson, L., "A New Five-Digit Multimeter That Can Test Itself," *Hewlett-Packard Journal,* August 1972.
5. Oliver, B. M., "Looking Ahead," *IEEE Spectrum,* November 1974.
6. Allan, R., "New Measurement Capabilities," *IEEE Spectrum,* November 1974.
7. Saba, Mona and Jack Grimes, "Microprocessors: A Component for all Seasons," *1974 Wescon Professional Program,* The Microprocessor Revolution, Part II.

Development of a portable compiler for industrial microcomputer systems

by LEROY H. ANDERSON
The Warner & Swasey Company
Cleveland, Ohio

WHAT ARE INDUSTRIAL MICROCOMPUTER SYSTEMS?

The development of the chip microprocessor in 1971 has enabled a revolution in the use of stored-program logic and data handling (the microcomputer) in industrial control applications not heretofore seriously considered.

According to a report on the Industrial Microcomputer Market,* published by the New York firm of Frost and Sullivan, "Microcomputers promise to be the fractional horsepower motor of the computer world. The market for them in industrial applications will explode from 7.9 million dollars in 1973 to 140 million in 1977 and 880 million by 1983." The Industrial Microcomputer System represents the next stage of development to the designer/user of industrial control equipment.

Microprocessors are general purpose digital circuits which can be programmed (with the addition of memory) for a particular users' requirements. Thus a microcomputer based system can more easily be changed or updated than an equivalent hardwired logic system as well as being able to perform arithmetic, logic and communication functions.

The Industrial Microcomputer System is capable of many of the same arithmetic, control and computational functions as a minicomputer, but at a fraction of the cost. The major differences between a microcomputer and a minicomputer based dedicated industrial control system are the cost, and the small physical size, the lower power drain and slower operating speed of the microcomputer. In addition, the mini may have a greater memory expansion capability.

However, in many dedicated industrial control applications the memory expansion capabilities and speed of the minicomputer are not necessary. Note that the microcomputer is by no means slow. As a matter of fact, the slowest microcomputer system usually can execute about 90 thousand instructions per second.

We, at Comstar, have been actively involved since 1971 in producing Industrial Microcomputer Systems. A typical industrial microcomputer system (shown in Figure 1) can be divided into five basic parts. They are: (1) the microprocessor and its associated memory, (2) the interface modules which connect the microcomputer system to external devices such as limit switches, push buttons or motor starters, (3) the equipment to program the microcomputer, (4) a program analyzer (shown in Figure 2) which is used to analyze and diagnose the operation of the microcomputer based system, and (5) a system tester (shown in Figure 3) which allows the user to check the microprocessor memory and interface modules to see if they are functioning properly.

To examine the characteristics of an industrial microcomputer system, let us start with the heart of that system—the Central Processor Unit (CPU) and its associated memories. The CPU performs all control and data processing functions.

Auxiliary to the CPU are the memories—usually both Random Access Memory (RAM) and Programmable Read-Only Memory (PROM). The PROM Memory is used to store the operating microcomputer programs or algorithms. The best type is erasable and electrically non-volatile. That is, it doesn't lose data if power is down.

Random Access Memory is used as a buffer storage to store data (such as intermediate variables used or printed during the process) which can be used in a volatile environment and stored in a volatile type memory. Random Access Memory may include I/O lines which may be used to drive input/output devices such as Light-Emitting-Diode displays.

Also available is Read/Only type Memory (ROM) which must be mask programmed and cannot be changed or altered, and Electrically Alterable Read/Only Memory (EAROM) which can be written and read back electrically and is electrically non-volatile.

The basic CPU plus the memories are then tied to some type of electrical interconnection (bus) system which connects the CPU to the memories and interface modules, allowing the input/output of information and/or control of peripheral equipment. Figure 4 illustrates a simple configuration.

Interface modules for an industrial microcomputer system can be broken down into the following categories; (1) digital modules, for digital communication with external devices using signals of 15 volts or less; (2) power switching modules, for driving power signals of up to 120 volts AC; (3) analog modules, which include analog-to-digital and digital-to-analog converters; (4) communication modules; both parallel and synchronous or asynchronous modules for serial communications, and (5) special

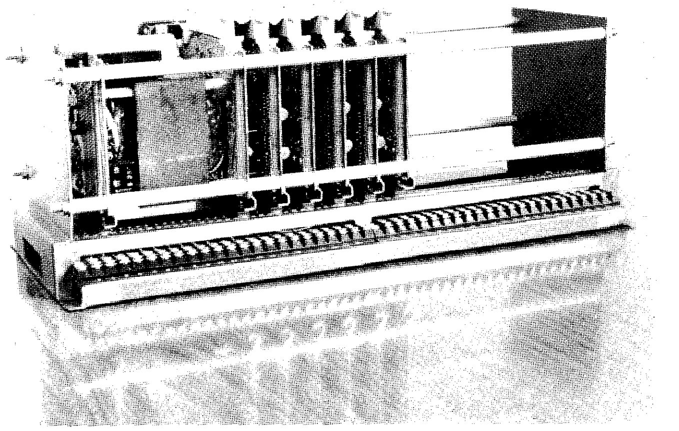


Figure 1—Typical industrial microcomputer system

modules, including pulse data modules, time-of-day clock, real time clock and watchdog timer, etc., all typically used in an industrial environment.

Peripheral equipment used with industrial microcomputer systems must be rugged, low in cost and very simple so it can have long life operation. The simplest types are push button and L.E.D. indicators. The next level of complexity is keyboards and alpha-numeric displays, usually gas plasma. At an additional level of complexity you start getting into small alpha-numeric printers and then into more sophisticated equipment such as floppy disc, magnetic tape cassette or even IBM compatible magnetic tape systems to store information for later data processing.

In the next few years there will probably be a whole new array of low cost peripheral equipment for industrial microcomputer systems because the processing section—CPU and memory—are becoming so cost effective that the user is being forced to analyze where he can obtain peripherals to match the performance and cost of the electronics.

Thus, to install industrial microcomputer systems you have to have a full range of memories, interface modules and peripheral equipment for the application. With this type of equipment, microcomputers can be used in the industrial computer control area, controlling test and assembly machines. They can be used in remote monitoring control for pollution monitoring, public utility control systems, waste water and monitoring systems. Simple machine tool control systems are now using microprocessors and, as the microcomputer becomes more powerful, probably all machine tool control systems will be microcomputer based. Many industrial data entry systems are based on microcomputers which preprocess the data and forward it to a central computer for sophisticated management information systems.

In addition, this same microcomputer system can be used for intersection traffic control, local traffic congestion control and traffic monitoring.

Also, modern material handling systems are using distributed microcomputers to control conveyors, packaging and palletizing equipment, as well as stacker cranes. Pallet moving equipment such as robot cars are used to completely automate product movement in a modern warehouse. Thus, the industrial microcomputers will be used for almost every control system where you need flexibility and future interconnect capability to large computer systems. In many cases, microcomputers now can even be cost effective against standard relay systems.

Figure 5 illustrates a microcomputer based industrial control system.

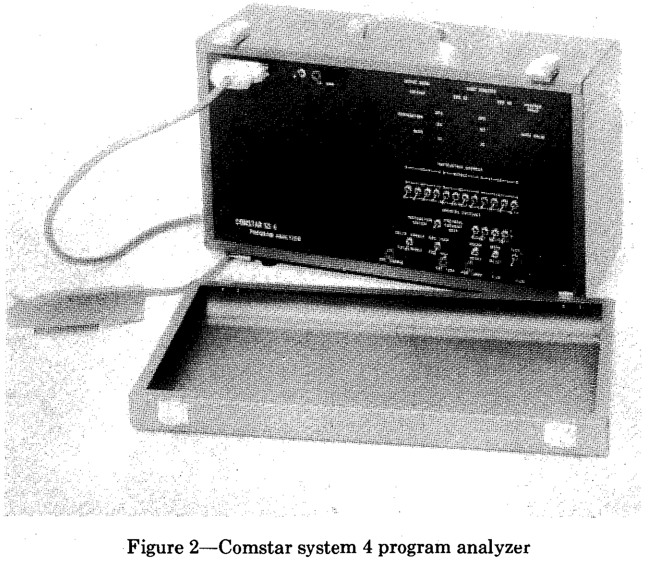


Figure 2—Comstar system 4 program analyzer

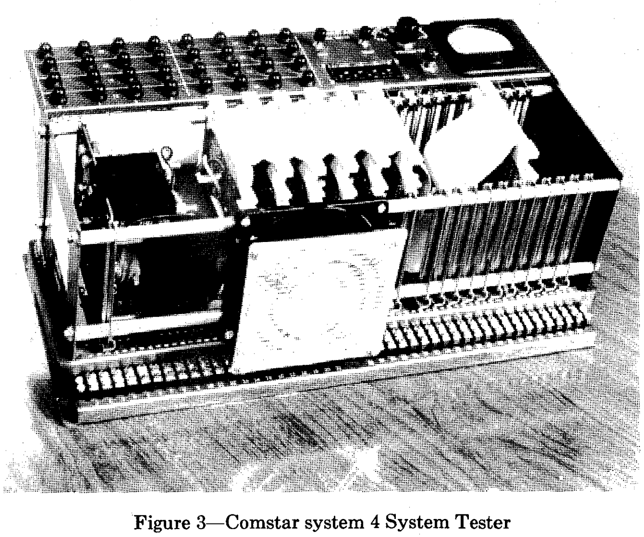


Figure 3—Comstar system 4 System Tester

A PROCESS CONTROL LANGUAGE (PCL)

Once an industrial control problem has been defined and a microcomputer system has been chosen to solve it, the programming becomes the next most serious problem. We at Comstar developed a Process Control Language (PCL) as an efficient method of programming industrial microcomputer systems. PCL closely resembles beginning FORTRAN in simplicity and is modeled after relay logic and common arithmetic and logic commands. This language allows a control program to be created in English, entered into a portable Process Control Compiler via a simple functional keyboard, and be converted into machine language which is stored in the Programmable Read/Only Memory (PROM) by the Compiler. PCL has reduced software costs by 50 percent or more.

PCL offers maximum capacity and flexibility for the professional programmer, but the non-programmer finds it easy to use and understand. PCL allows a Process Engineer to easily relay his program by way of the English language into the compiler and to create control algorithms stored in non-volatile PROM memory. This is an application oriented language and the unexperienced process engineer is able to quickly learn how to use it because it's conversational and relates directly to relay ladder charts and ladder diagrams and has Boolean algebra logic functions. The language also has bit memory and BCD character memory handling capability and it can store data changes on the inputs and outputs in Random Access Memory.

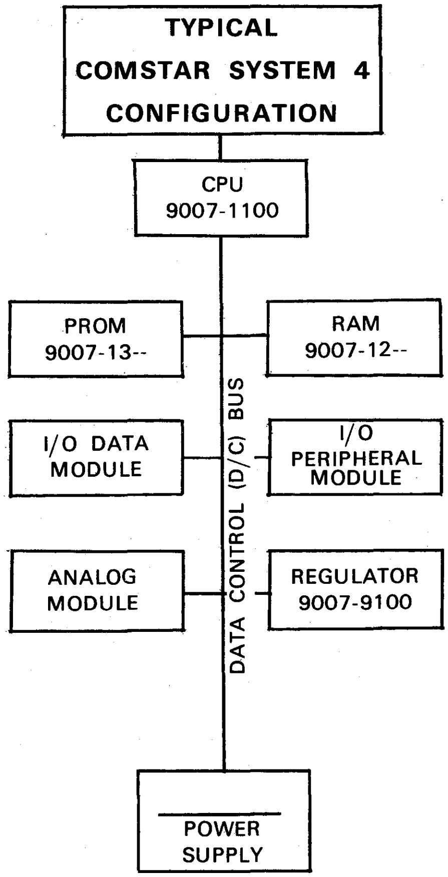


Figure 4—Simple Comstar microcomputer configuration

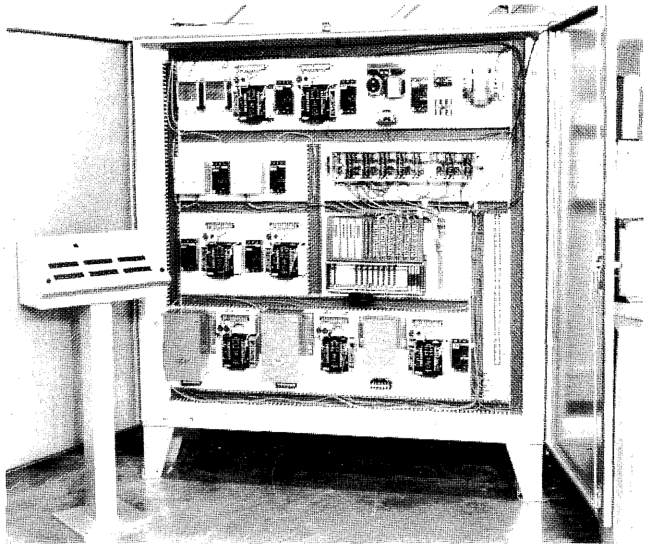


Figure 5—Microcomputer-based industrial control system

Freed from repetitious subroutine programming and confusing terminology, the Process Engineer can concentrate on using a repertoire of problem-oriented instructions that are designed specifically for the control project.

The PCL language provides uncommon flexibility. If a more efficient machine or system operating procedure is discovered, the engineer can readily change the control statements of his system to utilize the new development. If it's determined that certain types of system-operation information are needed, that data can often be retrieved through program modification. Adept programming can even allow the engineer to locate and, in some instances, by-pass malfunctions. And the entire control package can be made compatible with the user's present equipment and, if necessary, communicate with larger systems such as IBM's 360 series.

The simplest version of the Process Control Compiler can program a microcomputer with the following capacities:

Contact Closure Inputs	128 Contacts
Logic Power Inputs	128 Lines
Logic or Power Outputs	128 Lines
BCD Data Input	144 Digits
BCD Data Output	32 Digits
Bit Memory	192 Bits
Data Memory	256 Characters

Program Memory is expandable to 4096 8-bit words in 256 word increments, using up to 16 256×8 bit PROM chips. Data memory is expandable to 2560 4-bit words in 80 word increments, using up to 32 RAM memory chips.

Key-per-function instructions of the compiler

The Process Control Language offers the programming functions listed below and explained in detail in the appendix. Each is entered into the compiler via a single key.

1.0 Formatting
- 1.1 IF Causes execution of a statement if the value called upon by the operand is true.
- 1.2 AND Logical AND of two operands.
- 1.3 OR Logical OR of two operands.
- 1.4 THEN Logical transfer of an equation result.
- 1.5 NOT Causes operand to be checked for logical off.
- 1.6 END Indicates the end of a statement.
- 1.7 EOC (End of Compile)
  Flag to the compiler indicating end of compilation and return to entry and verification mode.

2.0 Status Testing
- 2.1 Memory XX
  Pointer to least significant digit of four BCD digit register.
- 2.2 MEM-BIT XX
  Pointer to flag bit in bit memory.

3.0 Input
- 3.1 INL XX
  Points to logical input line numbered XX.
- 3.2 IBCD XX NN
  Input of BCD data into memory addresses XX to NN, inclusive.

4.0 Output
- 4.1 OUT XX
  Points to logical output line numbered XX.
- 4.2 OFF-OUT XX
  Sets logical output line XX off.
- 4.3 OBCD XX NN
  Output of BCD data from memory addresses XX to NN, inclusive.

5.0 Data
- 5.1 ADD SS OO DD
  Add operands from 4-BCD-digit memory registers SS and OO, store result in memory register DD.
- 5.2 SUB SS OO DD
  Subtract 4-BCD-digit number in memory register 00 from the number in register SS and store result in memory register DD.
- 5.3 Compare: EQ, GT, LT
  Logical output if two operands are equal (EQ), the first is greater than (GT), or less than (LT) the second.
- 5.4 MOVE XXX YYY
  Move the contents of address XXX (1 BCD digit) to YYY.
- 5.5 LOAD XXD
  Load the value of D into BCD Memory location XX.
- 5.6 CLEAR XX
  Used to set a logical Memory bit to a zero.
- 5.7 SET XX
  Used to set a logical Memory bit to a one.

6.0 Branching
- 6.1 GO TO XXX
  Transfer control to statement XXX.
- 6.2 CALL XXX
  Transfer control to XXX (the address of a subroutine) and save the address of the next operand.

7.0 MACRO Instructions—Macro Instructions use the call instructions and pass parameters through memory.
- 7.1 Timing (0.1 Sec-999.9 Sec).
- 7.2 Counting (9999 per counter)—Tests multiple groups of up to 16 INL lines designated as counter inputs.
- 7.3 Analog Input—Inputs one of up to 128 channels of 8 or 4 digit BCD.
- 7.4 Analog Output—Outputs 3 or 4 digit BCD data onto one of eight channels.
- 7.5 Pulse Accumulation—Allows the PCL program to Reset, Restart, or Read the Pulse Accumulator or Frequency Monitor Module.
- 7.6 Time of Day Clock—Allows the PCL program to Read or Set the Time of Day Clock.
- 7.7 Quadrature Encoder Input—Allows the PCL program to read the Quadrature Encoder Input module for use in positioning systems.

In a later section, we will go through a typical application where a Process Engineer uses this type of language to program his industrial microcomputer system.

THE PORTABLE PROCESS CONTROL COMPILER

The process control compiler shown in Figure 6 is a small portable unit designed for programming the Comstar 4 Industrial Microcomputer System. The programming can be accomplished with high reliability even in field conditions. The PCL instructions are keyed in via the compiler keyboard, then converted into machine language and loaded into the PROM chip by the compiler. The input functions are displayed directly on a 32 character alpha-numeric plasma display, ensuring the user of a correct input. All keyed-in commands are stored in a buffer which can be verified with a key command. Up to 256 bytes of data or instructions can be entered. Data are compiled and can be dumped into a clean erased PROM chip. As an option, EIA or TTY outputs are available so the program can be printed out for future reference. A compiler can also edit, erase or program PROMS in machine language.

Future versions of this device will have expanded input and output capabilities and an external dual magnetic tape cassette terminal system to provide program storage and editing capability in the high level language without going to machine language. The system then functions almost like a remote terminal of a large time sharing system; but it is completely portable, demands no outside telephone connections and has proven to be extremely reliable as a portable programming device.

TYPICAL APPROACH TO USING PCL

The Process Engineer is the cornerstone to the use of PCL, of course, His first step is a detailed listing of all machine operations and their relationship to input and output devices in the system to be controlled or monitored. Operations must be divided into their basic individual steps.

Input and Output lines to all appropriate equipment in the control system are then assigned. A photocell sensor may require only one data line; whereas, a motor may require one for "motor forward" and another for "motor reverse." Counting and timing operations are then listed, and counters and timers are assigned.

The engineer then begins to write equations defining and controlling the operation of the system. Logic equations must be assigned to each operation listed in the first step. These equations are then grouped in a logical relationship and then entered into the compiler via the keyboard.

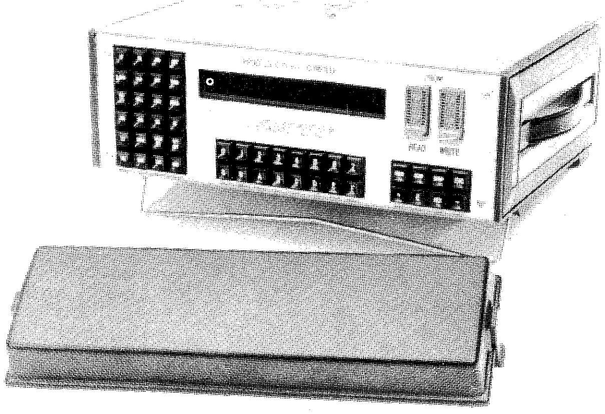
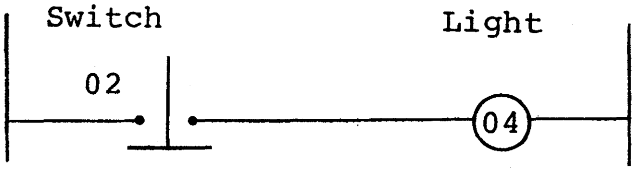


Figure 6—Comstar system 4 process control compiler

Sometimes it is helpful to the Process Engineer to use ladder diagrams as a tool when first learning PCL. For example, suppose he wishes to control a light with a switch. When the switch is up the light is on and when the switch is down the light is off. The ladder diagram would be:



The program would be:

Equations:

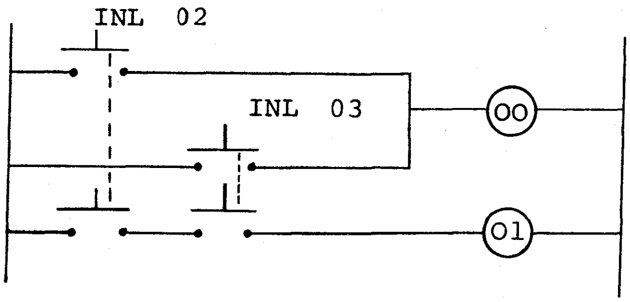
```
100 1F INL 02 THEN OUT 04 END
10E 1F INL 02 NOT THEN OUT-OFF 04 END
11D GOTO 100 END
11F EOC
```

Comments:

Statement No.	*Equation*
100	IF SWITCH 02 IS UP THEN LIGHT 04 IS ON
10D	IF SWITCH 02 IS DOWN THEN LIGHT 04 IS OFF
11D	CONTINUE CHECKING SWITCH
11F	END OF COMPILE

Consider another example where the problem is to turn on light 00 if either INL 02 or INL 03 is up. Turn on light 01 if both INL 02 and INL 03 are up.

Ladder Diagram:



Equations:

```
010 IF INL 02 OR INL 03 THEN OUT
    00 END
024 IF INL 02 AND INL 03 THEN OUT
    01 END
038 IF INL 02 NOT AND INL 03 NOT
    THEN OUT-OFF 00 END
```

Because we were designing the programming system for intelligent, but, in most cases completely inexperienced personnel, we wanted to make it as fail-safe as possible. In doing so, we developed the Compiler, the Analyzer, and a self-teaching Educator to provide a system of checks and balances that permits attentive persons of virtually any background to turn out accurate, functional programs after literally a few *hours* of instruction.

SUMMARY AND FUTURE

The development of a portable high level language compiler to be used with an industrial microcomputer system represents a major breakthrough in allowing Process Engineers to implement well designed control systems quickly and economically. The use of separate units to program, analyze and test the microcomputer system allows low cost, uncomplicated design of the actual controller. Because one compiler, analyzer and tester can be used to support many microcomputer controllers, they can be designed to be rugged, reliable and easy to use, without making the use of microcomputer control systems cost prohibitive.

The development of all successful computer systems is hinged on having well developed, easy to use software such as Fortran for scientific processing and Cobol for modern business processing. The Process Control Language that Comstar Microcomputers developed was based on Comstar's experience in installing almost a thousand microcomputer systems. The system was successfully developed by relating with Process Engineers in industry to determine what the real processing engineering needs were and what problems the engineer had to solve to successfully implement industrial microcomputers. Process Control Language and the Portable Compiler are some of the tools Comstar provides to help the Process Engineer solve those problems.

APPENDIX

Key-per-function instructions of the compiler

1.0 Formatting Instructions

1.1 IF Operand 1
Causes execution of a statement if the value called upon by operand one is true. Operand 1 may be MEM-BIT XX, Memory XX or INL XX where XX is a 2 digit hexadecimal number.

1.2 AND Operand 1
The result of the previous operand and the value called by Operand 1 are logically ANDed together. Operand 1 may be INL XX or MEM-BIT XX where XX is a 2 digit hexadecimal number.

1.3 OR Operand 1
The value called upon by Operand 1 is logically ORed with the value of the previous operand. Operand 1 may be INL XX or MEM-BIT where XX is a 2 digit hexadecimal number.

1.4 THEN
The THEN Operand allows the logical transfer of an equation result. For example, in the equation, 010 IF INL 02 AND INL OE THEN OUT 01 END, output line 01 would be set on if INL 02 and INL OE are both true. If the equation were false, control would transfer to the next operand after END.

1.5 NOT
The NOT pseudo-mnemonic may be appended to INL XX to form INL XX NOT or to MEM-BIT XX to form MEM-BIT XX NOT. The operand is checked for logically off when NOT is appended.

1.6 END
The END instruction must be the last operand of any IF statement, the last operand of any program, and must precede a statement number with the exception of the first statement number.

1.7 EOC
The End of Compile (EOC) instruction is a flag to the Compiler to indicate the end of the program segment stored in RAM memory. When the Compiler encounters an EOC it stops compiling and returns to the entry and verification mode.

2.0 Status Testing

2.1 MEMory XX
MEMory XX points to the least significant digit of a register that is four BCD digits in length. XX is a 2 digit hexadecimal number in the range OO-FF.

2.2 MEM-BIT XX
MEM-BIT XX points to a flag bit in bit memory, number XX. XX is a 2 digit hexadecimal number in the range OO-BF.

3.0 Input

3.1 INL XX
INL XX points to a logical input line numbered XX (INL 30=input line 30).

3.2 IBCD XX NN
IBCD XX NN is used for inputting BCD data, for example, from thumbwheel switches. XX is the starting number and NN is the ending number where the difference between them cannot be greater than F.

4.0 Output

4.1 OUT XX
OUT XX points to a logical output line numbered XX (OUT 3E=output line number 3E).

4.2 OFF-OUT XX Same as OUT except the output line is set OFF.

4.3 OBCD XX NN
OBCD XX NN outputs BCD numbers. Rules of usage are the same as IBCD.

5.0 Data

5.1 ADD SS OO DD
ADD OO to SS (BCD ADD) and store the results in DD. The operands point to memory registers which are 4 BCD digits in length. SS, OO, and DD are 2 hexadecimal digits in length.

5.2 SUB SS .OO DD
Subtract OO from SS and store the results in DD (BCD SUBtract). The operands point to memory registers which are 4 BCD digits in length. SS, OO, and DD are 2 hexadecimal digits in length.

5.3 Compare EQ, GT, LT
Compare is used in the following manner: *010 IF MEMORY 02 EQ MEMORY 03 THEN OUT OA END* Memory 02 is compared against Memory 03 for equality. Memory 02 and Memory 03 are 4 BCD digits in length. LT and GT are used in the same manner except a LT (less than) or GT (greater than) condition is checked for.

5.4 MOVE XXX YYY
The contents of memory location XXX are moved to YYY. The contents of XXX and YYY are 1 BCD digit in length, XXX and YYY are 3 hexadecimal digits in length.

5.5 LOAD XXD
The contents of BCD memory location XX is set to the value of D. XX is 2 hexadecimal digits in length and D may be any value from O to F.

5.6 CLEAR
The CLEAR operand is used for clearing Bit-Memory. It sets logical MEM-BIT XX to a zero (logical off).

5.7 SET XX
The SET operand is used for setting Bit-Memory. It will set logical MEM-BIT XX to a one (logical on).

6.0 Branching

6.1 GO TO XXX
Control is transferred to XXX, where XXX is a statement number 3 digits in length.

6.2 CALL XXX
Control is transferred to XXX address to a subroutine and the address of the next operand is saved. When the subroutine has completed execution, control must be transferred by a GO TO 7Cl. This will provide the necessary linkage for returning to the operand in sequence after the CALL XXX. If the Call is to a machine language subroutine then all that is necessary is a BBL instruction in the machine language subroutine.

7.0 MACRO Instructions
Macro Instructions use the CALL instructions and pass parameters through memory. These parameters are placed in memory by the PCL Program. Each routine will have dedicated locations in memory for its parameters.

7.1 Timing MACRO
The standard time base for the system will be a 100 millisecond square wave generated by a Real Time Clock and Control Module. This MACRO will set a MEM-BIT which is tested from the PCL Program and is also returned for testing with the IF instruction.

7.2 Counting MACRO
Groups of up to 16 INL lines (Range 00-7F) can be designated as counter inputs. For each counter input there will be assigned two Mem-Bits, one for system use and one to be tested by the PCL Program. INL lines that are designated as counter inputs can still be tested using the IF statement. Multiple groups of 16 can be assigned up to the limit of 128 total accumulations.

7.3 Analog Input MACRO
The Analog Input MACRO instruction will input one of up to 128 analog points of either 3 or 4 BCD digits of precision. These analog inputs will be stored in BCD Scratch Memory for access by the PCL Program.

7.4 Analog Output MACRO
The Analog Output MACRO instruction will output from BCD Scratch Memory onto one of eight (expandable with special software to 128 channels) channels in either 3 or 4 BCD digits of precision.

7.5 Pulse Accumulator or Frequency Monitor MACRO
This MACRO will allow the PCL Program to Reset, Restart, or Read the Pulse Accumulator or Frequency Monitor (PAFM) Module. Each of these modules contains two channels of PAFM. The data from the PAFM module would be stored in the BCD Scratch Memory. Multiple modules can be used up to a total of 5 channels at this time and up to 16 channels as a future option.

7.6 Time of Day Clock MACRO
This MACRO will allow the PCL Program to Set the Clock or Read the Clock. Data to be Set will be in Scratch Memory. Data read will be stored in Scratch Memory. The time of day clock module is capable of keeping time in either Centelis or Syderial time. System time displays are available using the OBCD command and 6 digit display.

7.7 Quadrature Encoder Input MACRO
This MACRO allows the PCL Program to read the Quadrature Encoder Input module for use in positioning systems. The BCD data is stored in the BCD Scratch Memory.

BIBLIOGRAPHY

1. Intel Corporation, *MCS-4 Microcomputer Set Users Manual,* Intel Publication MCS-042-374, March 1974.
2. Intel Corporation, *MCS-8 Microcomputer Set, 8008 8-Bit Parallel Central Processor Unit Users Manual,* Intel Publication MCS-056-0574, 1974.
3. Comstar Microcomputers, The Warner & Swasey Co., *Comstar System 4 Reference Manual,* Publication 0973-7020, September, 1973.
4. Comstar Microcomputers, *COMSTAR System 4 Process Control Language Programming Manual,* Publication 0574-7060, May 1974.
5. Comstar Microcomputers, *COMSTAR System 4 Compiler Hardware Reference Manual,* Publication 1173-7050, November, 1973.

Microprocessors in CRT terminals

by JOHN WHITING and SANDY NEWMAN
Beehive Medical Electronics
Salt Lake City, Utah

INTRODUCTION

The recent introduction of integrated circuit microprocessors has produced a new variety of CRT terminal—the firmware terminal. The firmware terminal incorporates a microprocessor to control data flow, using a control program supplied by the terminal manufacturer in Read-Only-Memory. Priced only slightly above hardwired editing terminals, it can perform far more complex functions, assisting both the operator and the computer system into which it is connected. It is priced well below the user-programmable terminal which requires magnetic storage for program and a much higher level of sophistication from both the sales force and users.

WHAT IS A CRT TERMINAL?

The CRT computer terminal is one of a variety of devices used to communicate with computers. It is distinguished by using a television screen (CRT) for presenting computer data to the human operator, rather than using an electric typewriter, loudspeaker, flashing billboard or other device. Messages appear on the screen as several lines of words, much as they would appear on a typewritten sheet of paper; a common format is 25 lines of 80 characters each, including spaces. Punctuation marks, numerals, and other symbols may appear as well as upper and usually lower case characters.

The operator inputs data through a keyboard much like a typewriter keyboard. Keyboard data may go directly to the computer, but more commonly is temporarily stored in the memory used for the display. Thus the message entered from the keyboard appears on the CRT screen just as it would if it were being typed on paper by a typewriter, but with several advantages. For one, changes can be made by simply backing up the cursor, which marks the point at which data is entered, to the point to be corrected, entering the correct data in place of the incorrect in the display memory, and then moving the cursor back to wherever data entry left off. For another, if the change requires inserting or deleting characters or words, the terminal can shift characters following the change either right or left to make additional room or close up space as necessary, even though this may involve moving entire words from one line to another. Then, since the message is stored as digital codes in the display memory, it can be transmitted to the computer on command without the need for converting printed characters with an optical reader.

The fundamental elements of a CRT terminal are shown in Figure 1. Data comes into the terminal either from the keyboard or from the serial port. The serial port sends and receives data to a computer mainframe as a serial stream of digital bits, at rates up to 1200 characters per second. (If the mainframe is located very far away, a device called a data set translates the digital data into audible tones to be sent over telephone lines to a second data set which translates the tones back into digital data.) The control logic reads/writes display memory data at the location defined by the cursor control. The cursor advances as characters are written and can also be moved about by control codes from either the keyboard or the computer. Separate character generation circuitry is used to directly access memory data and translate it into dot patterns to drive the CRT. The circuitry must rewrite the characters on the CRT 50 to 60 times a second to maintain a flicker-free display, so high-speed circuits are used for this function. Relatively slow-speed circuits can be used for the control logic which performs a far wider variety of functions, some of which may be very complex, depending on the application.

WHAT A MICROPROCESSOR CAN DO

A microprocessor with its associated program memory and interface circuits can be used very effectively in place of hardwired control logic to pass data between the keyboard, serial port, and display memory. In this position, it can intercept and interpret codes as they are received, storing characters in memory and executing control functions as necessary. It can also manipulate data in the display memory and format messages to send to the mainframe. Not only does this make feasible the execution of much more complex functions than possible with hardwired logic, it reduces the cost and time required to modify terminal functions for special applications. Since the hardware is unchanged by such program modifications, extensive retesting of the terminal design is not required, reliability and serviceability are unaffected, and all terminals of a given hardware type can be manufac-

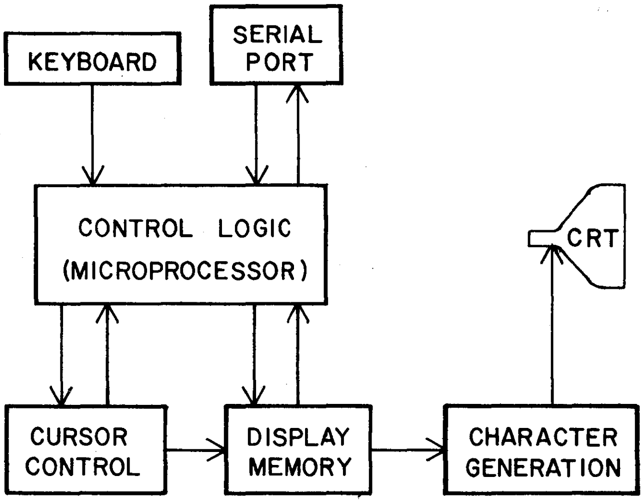


Figure 1—CRT terminal block diagram

tured on the same production line with the specialized program installed only after initial testing is completed. Thus, small quantity special orders can enjoy many of the mass production advantages of a standard product and yet have many specialized functional characteristics.

For example, a keyboard whose keys are coded to be teletypewriter compatible can be made typewriter compatible by changing a few keycap legends and translating the codes received from those keys with the microprocessor. Function keys can similarly be moved around on the keyboard by moving the corresponding addresses in the firmware table used to locate the function routine. Functional modifications are also straightforward. For example, a call to the Cursor Down subroutine can be inserted in the Carriage Return routine to make the Return key do both a Return and Line Feed, and the Skip-Cursor Right key sequence may be changed to move the cursor right by 20 positions instead of 16 by simply changing a numerical constant in the program.

The ability of the microprocessor to give specialized interpretation to both functional controls and displayable graphics originating from the keyboard also applies to codes sent from the central processor. Thus, the special codes which cause portions of the display to blink or to display black characters on a white background rather than white on black for emphasis may be transmitted over the communication lines as a two-code sequence but stored in the display memory as a single code. Code sequences may also be used for control functions and setting cursor positions, margins or tab locations, with the microprocessor translating numeric data between binary and decimal or other code formats.

In multi-drop polling networks the microprocessor can carry on interactive exchanges with the central processor. These can be very simple exchanges or quite complex, according to the requirements of the system. For example, a start-of-header code, an identifying address code, and a single status or command code may form the entire message, or the format can be expanded with preceding codes for word synchronization, multiple address and status codes, terminating codes and error check codes. The terminal might transmit only on operator command, or the central processor may be able to interrogate the terminal to find out whether the terminal is on-line or off-line, dumping data to a slave printer, actively entering information from keyboard, waiting to send data or waiting for a reply. Since the terminal program has complete control over message format, many different systems can all use the same terminal hardware.

The microprocessor can also control some of the communication signal lines directly; and when the interface is designed appropriately, it can be programmed to cooperate with other terminals in a serial string (daisy-chain) to a single data set or computer port for increased efficiency and reduced cost. As a general rule, the microprocessor should have as much control as possible over data flow and logic signals but timing controls such as Clear-to-Send delays are best handled by hardware which can be adjusted as necessary at each installation.

One of the most complex functions performed by the better hardwired terminals is the Delete Line, where a counter is employed to delete all 80 characters on a line of text; a microprocessor can use an internal register for the same purpose. Besides repetitive functions, such data-dependent functions as Delete Sentence can be performed, with the microprocessor deleting all characters between any two sentence delimiters. A variety of algorithms may be used to maintain word integrity and adjust column widths, giving special treatment to hyphens and other formatting codes. The microprocessor can search through the display memory data to locate all instances of whatever word the operator wants to find for correction or verification. In an accounting application, a column or row of numbers could be summed by even simple microprocessors, but more difficult calculations may be better handled by either the mainframe or a $20 pocket calculator. Other special functions use the microprocessor's access to the mainframe as well as to the display memory to take advantage of a centralized data base.

Terminal functions to be performed on command from the mainframe should generally be executable in one or two milliseconds at most, corresponding to the time required to transmit data codes over fast serial communication lines. Fortunately, the data manipulations which take long times for the terminal microprocessor to execute can be performed within the mainframe before the text is transmitted.

RAM VERSUS SHIFT REGISTER

The role of the cursor control circuitry depends on whether random-access memory (RAM) or shift register memory is used for the display memory. To write a character into a RAM memory simply requires addressing the matrix point and writing the code directly from the data bus, a natural task for a microprocessor. But to erase the entire memory with this direct approach requires sequentially addressing approximately 2000 memory locations and writing spaces in them, a very time consuming task for a microprocessor. Shift register memory is like a garden hose full of marbles; for each character stored at the input end of the memory, one character is pushed out at the output. When each character output is put back into the input, a memory is formed with sequential access to memory characters. To change a character, the new character must be put in the data stream in place of the former character at just the right point in the memory cycle, so the cursor circuitry must hold the new character when it is output from the data bus, and identify the right point in time to break it into the shift register data stream. But to erase the shift register memory, it is only necessary to stuff space characters into the input for one full cycle of the memory. Other tasks involving sequential access of many memory locations are insertion and deletion of characters in text, skipping over protected fields, tabulation by field, paragraph, or other text block and search for Start-of-Message characters. The abundance of such functions is a consequence of the sequential nature of the data displayed on the CRT, itself derived from the sequential nature of human thought. Thus shift registers are often used for display memories despite the awkwardness of writing single characters in them.

Both random and sequential access to the display memory is possible with RAM memory supported by specialized hardware for handling those sequential addressing functions which must be executed rapidly and with minimal microprocessor support. How much specialized hardware is required depends on what functions the terminal must perform faster than the microprocessor can do them. Such hardware can sequentially access currently available RAMs nearly as fast as popular shift registers, but adds to the cost and complexity of the terminal. Another alternative is the use of RAM memory, organized as many small record blocks linked by address characters. While this uses some memory locations for the linking and complicates display generation, it may be the most powerful and versatile scheme for an adequate microprocessor. With the low cost RAMs and better microprocessors recently made available, we expect to see changes from the traditional use of shift registers in terminals.

CHOICE OF A MICROPROCESSOR

The speed required of a microprocessor to be used in a CRT terminal depends largely upon the interface between it and the display memory, and the demands of the application. Conventional measures of processor speed, such as time to add two numbers, are not particularly meaningful here, as long as the microprocessor can identify and respond to approximately 1200 codes per second, which even the primitive 8008-1 microprocessor from Intel can do (just barely). With a shift register memory, throughput is usually limited by the time required to read characters as the cursor is moved through memory. With a RAM memory the limitation is in sequential access tasks, such as selectively erasing flagged data, unless either hardware support is given to perform all required sequential addressing tasks, or the terminal specification can be written around the microprocessor's limitations. (Since firmware CRT terminals are usually sold as improvements over hardwired, shift register memory CRT terminals, fast execution of sequential tasks is generally assumed by salesmen and customers alike.) Given adequate hardware support, almost any modern microprocessor is fast enough; without hardware support, look for microcoded instructions to repeatedly read-swap-write and index in approximately one microsecond.

CRT terminals are very price-competitive, and the cost of Read-Only-Memory for program storage invariably exceeds the cost of the microprocessor itself, with Programmable ROMs costing many times more. Since memory cost is included in sales price, the instruction set should be optimized for minimum memory utilization. The Exclusive-OR instructions of the 8008-1 are very nice for computing check characters and the even-odd parity flag is used in the most speed-critical routines. Other features which we have found useful are the logical and arithmetic operations with the following 8-bit byte, and especially the compare-immediate instruction. Since almost all routines executed in a CRT terminal application are simple and short, conditional branch and subroutine call instructions find heavy usage and relative addressing is most convenient. Subroutine nesting goes four or five levels deep at most and a LIFO stack for the program counter is very helpful.

Interrupts are not needed as long as the terminal is doing just one task at a time, as is often the case; however, in polling systems where both the operator and the central processor are sending data simultaneously, a reasonable interrupt handling capability is almost essential. This requirement also arises if the terminal is to both send and receive data simultaneously, or is to pass data from the central processor to a slave printer while continuing normal keyboard service. Thus the terminal application as well as the hardware design affect the choice of a microprocessor from among the many available.

WRITING TERMINAL CONTROL PROGRAMS

The need for specialized hardware to debug programs and program PROMs confines the writing of programs to a few specialists at the factory and sophisticated OEM customers. Consequently, machine language is used and description of the hardware-software interface is largely word-of-mouth between the hardware designers and the programmers, which makes it even more difficult for outsiders to do their own programming.

The first step in modifying a program is understanding what is required and how the terminal is to be used. Then the routines which are affected by the change must be identified which presumes knowledge of the methods and structure of the program. Register usage in particular

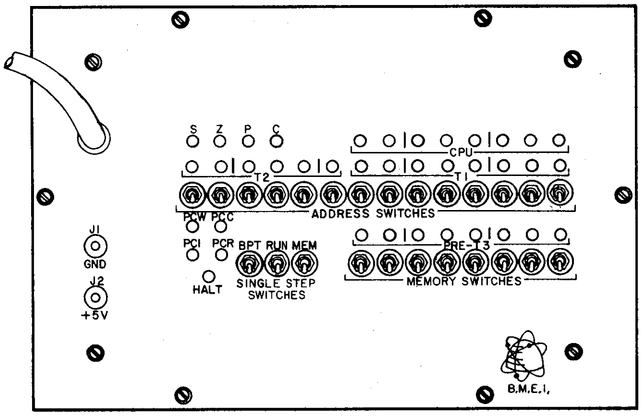


Figure 2—8008 microprocessor control panel

must be understood, since temporary data cannot be stored in PROM memory and scratchpad space is limited. Familiarity with the hardware devices to be used in execution of the function is presumed, including worst case timing for access to hardware indicators following a change of state. Following coding of the change, it must be debugged and some means of testing it provided, usually at the customer's site. Finally, the change must be documented so that it can be duplicated or modified in the future.

DYNAMIC DEBUG

The problem of dynamic debugging is complicated by the fact that the microprocessor has no independent status indicators other than the display of the terminal being debugged. Three solutions to this problem are: replace the microprocessor chip directly in the printed circuit board with a black box version of the microprocessor incorporating a control panel; simulate the microprocessor on a minicomputer; or write a program debug to be resident in memory alongside the terminal controlling program.

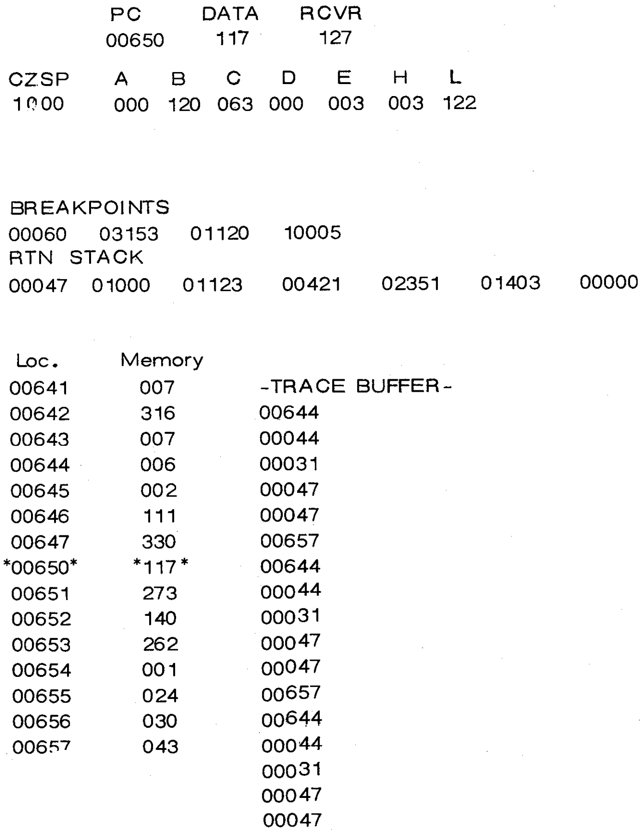
Microprocessor control panel

Figure 2 shows the control panel of a black box used in place of an Intel 8008 microprocessor. It plugs into the microprocessor socket through an umbilical cord and contains an 8008 chip interfaced with latches and timing circuits to permit either single stepping of instructions or normal execution to a breakpoint address. LED's indicate address and bus data as well as machine cycle type and status flags. The breakpoint address is specified by toggle switches, as is a data byte which may be substituted for bus data. This substitution capability is very useful in debugging hardware with repeated execution of an instruction, as well as in correcting instructions or modifying data while single stepping through a program. The main drawbacks of the black box approach are that substitutions must be inserted by hand each time the program encounters the bad byte, and only one breakpoint can be set at a time. The advantages are the real-time execution, straightforward single-stepping, economy, portability and universality of application in any 8008 system.

Simulator

Execution of a microprocessor instruction set can be easily simulated within a minicomputer by interpreting each program step and imitating its execution. However, input and output instructions which control a terminal's internal devices can best be simulated by passing these instructions to a terminal programmed to recognize the command, cause it to be executed and reply with an acknowledgment which includes data from input instructions. This combines the power of minicomputer simulation and normal control of keyboard, display, and other internal devices, except for the serial transmitter and receiver. The disadvantage of this approach is the slow simulation speed of I/O instructions, since five serial codes must be passed for every I/O instruction executed.

The console display of a simulator for a Super Bee terminal using the Intel 8008 microprocessor is represented in Figure 3. It shows a program halted at location 650 prior to executing a 117 (INPUT device 7) as indicated by



```
             PC        DATA      RCVR
            00650       117       127

CZSP        A    B    C    D    E    H    L
1000       000  120  063  000  003  003  122


BREAKPOINTS
00060   03153   01120    10005
RTN  STACK
00047  01000   01123    00421    02351    01403    00000


 Loc.       Memory
00641        007        -TRACE BUFFER-
00642        316        00644
00643        007        00044
00644        006        00031
00645        002        00047
00646        111        00047
00647        330        00657
*00650*     *117*       00644
00651        273        00044
00652        140        00031
00653        262        00047
00654        001        00047
00655        024        00657
00656        030        00644
00657        043        00044
                        00031
                        00047
                        00047
```

Figure 3—Microprocessor simulator display

the Data field. The condition flags set by the last arithmetic operation are all low (not zero, no sign and odd parity) except for the carry indicator, which is high. The contents of each register, A, B, C, D, E, H, and L are as shown on line 4. Four breakpoints have been set at 060, 3153, 1120 and 10005, which halt execution and cause the console display to be regenerated if they are encountered during execution. The RTN stack is a last in first out (LIFO) stack of return addresses. The contents of memory from 641 to 657 are displayed. The trace buffer records the branches of a program. Given is an example of repeated execution of a short service loop. The last jump was to location 644 (top of the stack) and originated from location 044. The program arrived at 044 by sequential execution from 031. The program jumped from location 047 to 031 and arrived at 047 by returning from 657. The top of RTN stack indicates that the next return will also be to location 047, repeating the sequence as seen below in the trace buffer.

Each of the items on the console display may be modified from the console keyboard at any point in the execution of the program. Additional keyboard functions available by single key depressions at the console are Single Step, Program Run, Examine Contents at PC, Modify Contents at PC, Read Paper Tape, Examine Trace Buffer, and Punch Paper Tape. A control "T" from the console may be used to halt program execution. The Carry switch on the minicomputer control panel causes characters from the console keyboard to be interpreted as serial inputs to the terminal as if from a mainframe and are shown in the RCVR field of Figure 3.

This system eliminates the nuisance of erasing and reprogramming PROMs for program development, since the program stored in the minicomputer memory can be easily modified from the console. Overlay techniques can be used to modify blocks of programs, which is particularly useful in changing routines and polling protocols when isolated as stand-alone items. The addition of peripheral memory allows a program to be assembled, executed, modified, debugged and punched into PROM programming format without leaving the control console. Of course, this is not a portable system and the long execution time of I/O instructions (about 400 times normal) restricts testing of functions whose speed and efficiency depend on multiple display memory operations. However, the slow execution enhances visualization of complex or extended display functions.

Resident debug

If memory space is available in the terminal, a resident debug program may be written which can either single step or run the terminal control program to breakpoints. This debug program must share the CRT display with the terminal program, either by allocating a portion of the screen to the debug program or by swapping terminal data in and out of auxiliary RAM memory to make room for the debug display. When the same RAM is used for storage of the terminal program, substitutions can be quickly and easily made and breakpoints may be inserted to return program control to the debug program; but means must then be provided to load the terminal program into the RAM from the assembly system. Register contents, memory contents, program counter, breakpoints, return stack and status flags can be displayed and modified if the necessary Push and Pop instructions are available (as on the Intel 8080, but not the 8008). Care should be taken to avoid placing breakpoints or starting execution in the middle of the multi-byte instructions. Such a debugging system has the advantages of executing a self-contained program at real-time speeds with normal access to I/O devices, albeit in a modified terminal.

SUMMARY

We expect to see accelerated growth in the utilization of firmware CRT terminals as users become more familiar with their advantages and capabilities. By taking advantage of terminal capabilities in systems design and assigning to the terminal those tasks which it can best perform, system throughput can be maximized, while operator-related functions can enjoy the flattery of dedicated processing.

Designing an application oriented terminal

by J. P. KOHLI
NCR Corporation
Dayton, Ohio

INTRODUCTION

Application oriented Terminals have been around for quite some time and traditionally they have been hard wired. The approach, however, has been changing recently due to the availability of low cost RAM memory and microprocessors. The main reason for the trend is that although the hard wired Terminals cost less, they do not provide or have the flexibility a customer is provided with a programmable Terminal. However, the cost of programming becomes an important consideration and will vary depending upon what data is available with the Terminal to aid the programmer. In designing the Banking Terminal, Honeywell used a combination of the traditional hard wired implementation approach and the complete programmability approach. Honeywell also provided for a COBOL type "FITAL" (**F**inancial **T**erminal **A**pplication **L**anguage) user level language for the customer, to aid programming the transaction sequences which allowed the customer to reduce programming cost. The limited programmability approach reduces the amount of programming, while it does not sacrifice flexibility available to the customer to tailor the Terminal transactions to suit his requirements.

This paper presents the Honeywell design approach by describing:

1. Bank Teller Terminal 7340 and its features.
2. Operation of the Terminal.
3. Firmware/RAM Architecture of the Terminal.
4. Real Time Processing Considerations in the Firmware Design.
5. Customer Programming in RAM.
6. Programming Aid to the Customer (FITAL Language).
7. Conclusions—Advantages and disadvantages of using the Honeywell approach versus hard wired or completely programmable terminal design approach.

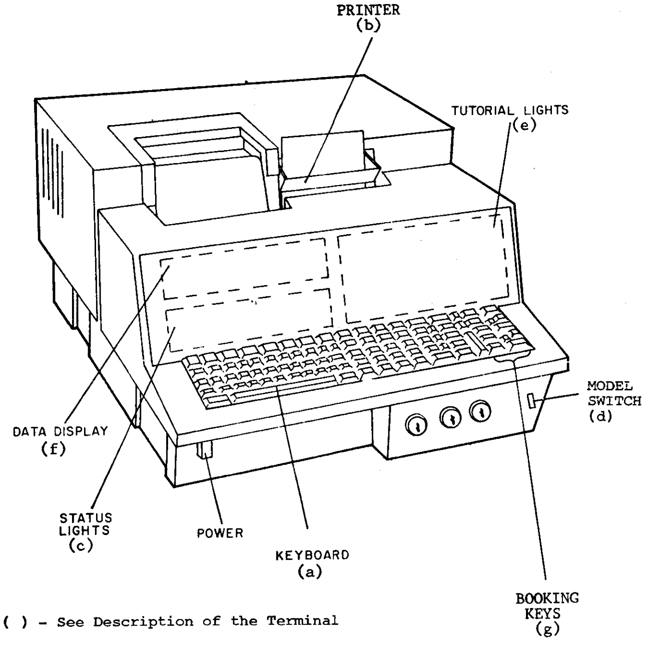
TERMINAL DESCRIPTION

Bank Teller Terminal (BTT 7340) is a data entry device designed to be located in a teller window to record and control transactions in a banking environment. The Terminal logs transactions and prints on a passbook and a journal through real time communications to a computer, thus aiding the teller in accepting and processing transactions. The transaction sequence is customer programmable and is loaded in the Terminal memory through the communication line via the host processing facility. The Terminal has on-line/off-line diagnostic modes. It is a self-contained unit and does not require a separate controller. Some of the salient features are described below and shown in Figure 1.

(a) *Keyboard*—Used to enter transactions from the Terminal. Includes alpha, numeric, and programmable functional key capabilities.
(b) *Printer*—160 column printer is used for printing the Journal (audit trail of transactions), passbook, and customer receipt.
(c) *Status Lights*—Ten indicator lights that indicate the status of the Terminal and the communication line at any given time.
(d) *Mode Switch*—Used to control the Terminal mode (On-line, Off-line, or Diagnostic mode).
(e) *Tutorial Lights*—Up to 28 programmable indicator lights that can be interlaced with the transaction sequence providing tutorial lead-through to the teller and/or providing a pictorial history of keyboard action while entering a transaction.
(f) *Dynamic Data Display*—A 32 character alpha/numeric display allows display of keyboard entry and editing and provides up to eight 32 character lines for forms fill-out or Inquiry display via communication messages.
(g) *Booking Keys*—Provide for teller identification when booking the Terminal.

TERMINAL OPERATION

During power turn on a sequence of hardware reliability checks are performed automatically. If the checks pass, the Terminal is ready to accept the user program tables via communications with the central processor or a back up Load Cassette Unit. Once the tables parameters are loaded, the table processor in the Terminal ROM memory processes the tables. Each table contains information for illuminating tutorial lights, performing specific terminal functions and address of the next table to be processed.



( ) - See Description of the Terminal

Figure 1—Bank teller terminal—7340

There are twenty-four different types of tables available to the user. To aid the user, a COBOL type language "FITAL" is available for programming the various terminal functions.

TERMINAL FIRMWARE/RAM ARCHITECTURE

The heart of the Terminal is the Basic Logic Unit (BLU), a Honeywell developed microprocessor which has been used in a number of other Honeywell Terminals. Figure 2 shows the overall architecture of the Terminal. The hard wired programming in ROM (firmware), the customer programming of loadable list processing type tables in RAM, the standard interrupt and Input/Output interface of the microprocessor, through the Device Oriented Electronics (DOE), help drive the Terminal. The logic to process the customer tables and specific processes unique to the Terminal are in the ROM; the customer programmable parameter tables are loaded in RAM. In addition to the tables, subroutines may be written in BLU machine language to achieve functions beyond those provided by the tables. For the microprocessor, ROM, and RAM are interchangeable and, therefore, the question becomes one of which functions of the Terminal should be in ROM and which functions should be stored in RAM. Obviously programming the Terminal functions in ROM is cheaper but not flexible; whereas providing them through user parameter tables is more expensive but allows the user control of the transaction sequences per his needs. Therefore, the major consideration in deciding between ROM and RAM was that the customer should be able to program the Terminal per his requirements with a minimum of programming. Subsequently, all the Ter-

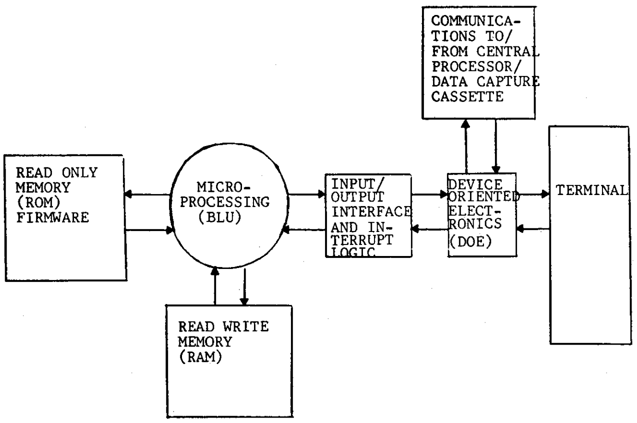


Figure 2—Firmware/RAM architecture of the terminal—BTT 7340

1. Communications control procedure for interface with the Host Processor.
2. Loader and processor for the RAM tables.
3. Various Terminal Input/Output functions; i.e., open passbook door, print passbook, buffering keyboard inputs, interface Data Display, etc.
4. Reliability checks and test & diagnostics for the Terminal.
5. Add routine to perform various adding, subtracting functions.
6. Reset capability for the Terminal.
7. Auto insertion of decimal point and right justified printing of amount fields.

Figure 3—Functions programmed in ROM

1. Transaction Sequences and Responses
2. Functioning of Tutorial Lights during Transactions
3. Formats for Journal, Passbook, Receipt and Validation Printing
4. Ability to Override Terminal Messages from the CPU
5. Ability to use Mode Switch, Function and Booking Keys in a desired way
6. Ability to use Customized Account Number Verifying Routine (check digit)
7. Ability to Load and Call Special Routines
8. Interpretation of Function Keys
9. Specifying Fields as Alpha/Numeric or Numeric Only
10. Checking of Acceptable Field Length

Figure 4—Functions available in RAM through programmable tables

minal features that are fixed and unique to the Terminal (For Example: Opening the passbook door, moving the carriage, causing the print head to operate, communicating with the processor, test and diagnostics for Terminal, etc.) are programmed in ROM. However, controlling the sequence of steps within a transaction, formats of printing on passbook and Journal, selected illumination of tutorial lamps and interpretation of function key codes, etc., are considered programmable and are stored in RAM by the user program. These programmable functions are provided to the customer by list processing type program tables. There is a table processor in ROM that processes these RAM tables after the customer has loaded them into the Terminal's memory. The customer programs these tables per his banking environment using "FITAL" language and associated compiler and then loads them into the Terminal's memory through the use of a loader (also programmed in ROM). Figure 3 shows the various Terminal functions programmed in ROM. Figure 4 lists the Terminal functions available to the customer through the use of the program tables.

REAL TIME PROCESSING CONSIDERATIONS IN THE FIRMWARE DESIGN

The Terminal is designed to communicate with the host processor during transaction processing in real time. In addition, while communicating with the host processor, keyboard data may be entered and the printer may be activated. Since these functions involve simultaneity of operations for the Terminal, a mini-operating system in firmware was designed to handle these independent parallel Terminal operations. The design includes a central executive which processes five subexecutives in partitional segments as shown in Figure 5. The choice of five subexecutives and the amount of partitional segments to be executed each time, was dictated by parallelism in operations and the speed of operations; e.g., communication line can communicate up to 4800 baud synchronous, printer can print characters at the rate of 20 characters/second, the keyboard data can be entered at 10 characters/second.

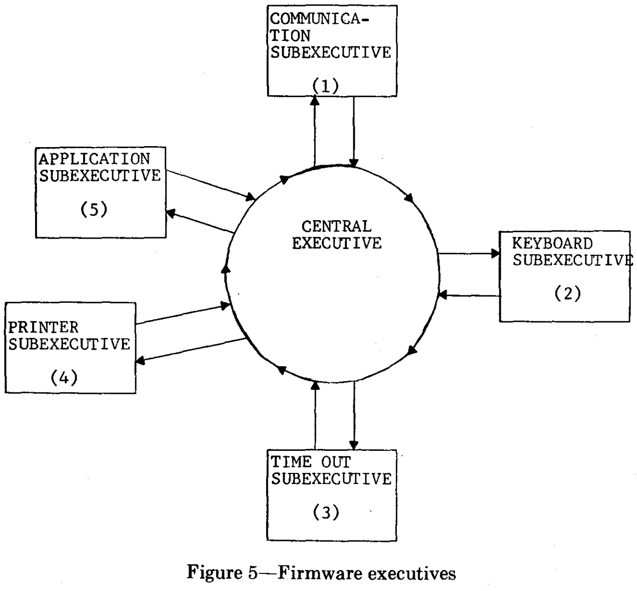

COMMUNICA-TION SUBEXECUTIVE (1)
KEYBOARD SUBEXECUTIVE (2)
TIME OUT SUBEXECUTIVE (3)
PRINTER SUBEXECUTIVE (4)
APPLICATION SUBEXECUTIVE (5)
CENTRAL EXECUTIVE

Figure 5—Firmware executives

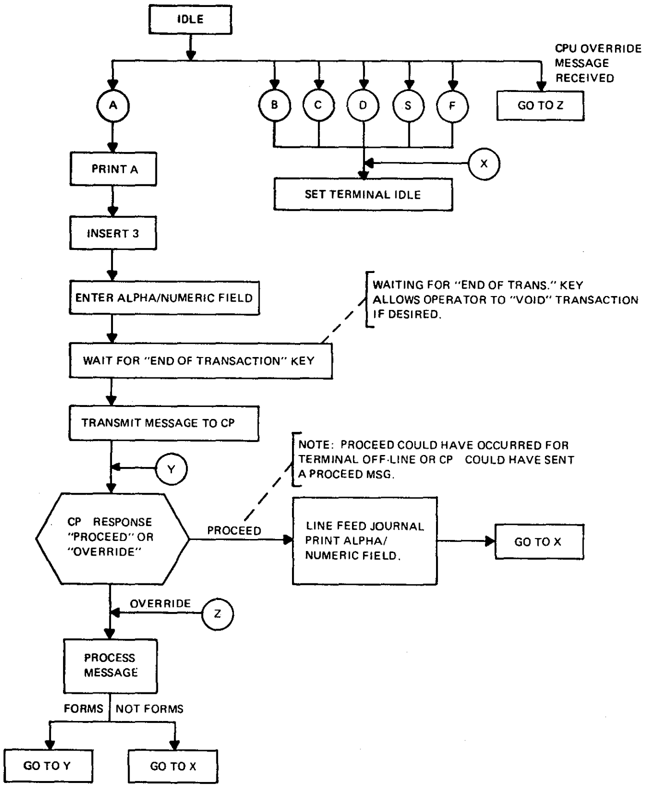

IDLE
CPU OVERRIDE MESSAGE RECEIVED
A
B
C
D
S
F
GO TO Z
PRINT A
X
SET TERMINAL IDLE
INSERT 3
ENTER ALPHA/NUMERIC FIELD
WAITING FOR "END OF TRANS." KEY ALLOWS OPERATOR TO "VOID" TRANSACTION IF DESIRED.
WAIT FOR "END OF TRANSACTION" KEY
TRANSMIT MESSAGE TO CP
NOTE: PROCEED COULD HAVE OCCURRED FOR TERMINAL OFF-LINE OR CP COULD HAVE SENT A PROCEED MSG.
Y
CP RESPONSE "PROCEED" OR "OVERRIDE"
PROCEED
LINE FEED JOURNAL PRINT ALPHA/ NUMERIC FIELD.
GO TO X
OVERRIDE
Z
PROCESS MESSAGE
FORMS
NOT FORMS
GO TO Y
GO TO X

Figure 6—Transaction Flow Chart Example

Functions that different subexecutives perform are as follows:

1. *Communication Executive*
   Performs the loader function for loading customer programmed tables in RAM and provides for communication to and from the host processor.
2. *Keyboard Executive*
   Performs keyboard functions of receiving and storing keyboard data to be processed by the application executive.
3. *Time Out Executive*
   This executive continuously checks for the expiration of various time outs that may be set by printer or application executives. It signals the completion of the time out to the other executives.
4. *Printer Executive*
   Performs all the printer functions of printing on the Journal and the passbook.
5. *Application Executive*
   Performs the rest of the execution for the Terminal

which includes:

(a) Processing the customer program tables in RAM.
(b) Performing various terminal activities such as passbook control functions, processing data received from communication and keyboard executives, providing data to the printer executive, displaying terminal status, etc.
(c) Test and diagnostics for the Terminal.
(d) Power on initialization and reliability checks on the Terminal.

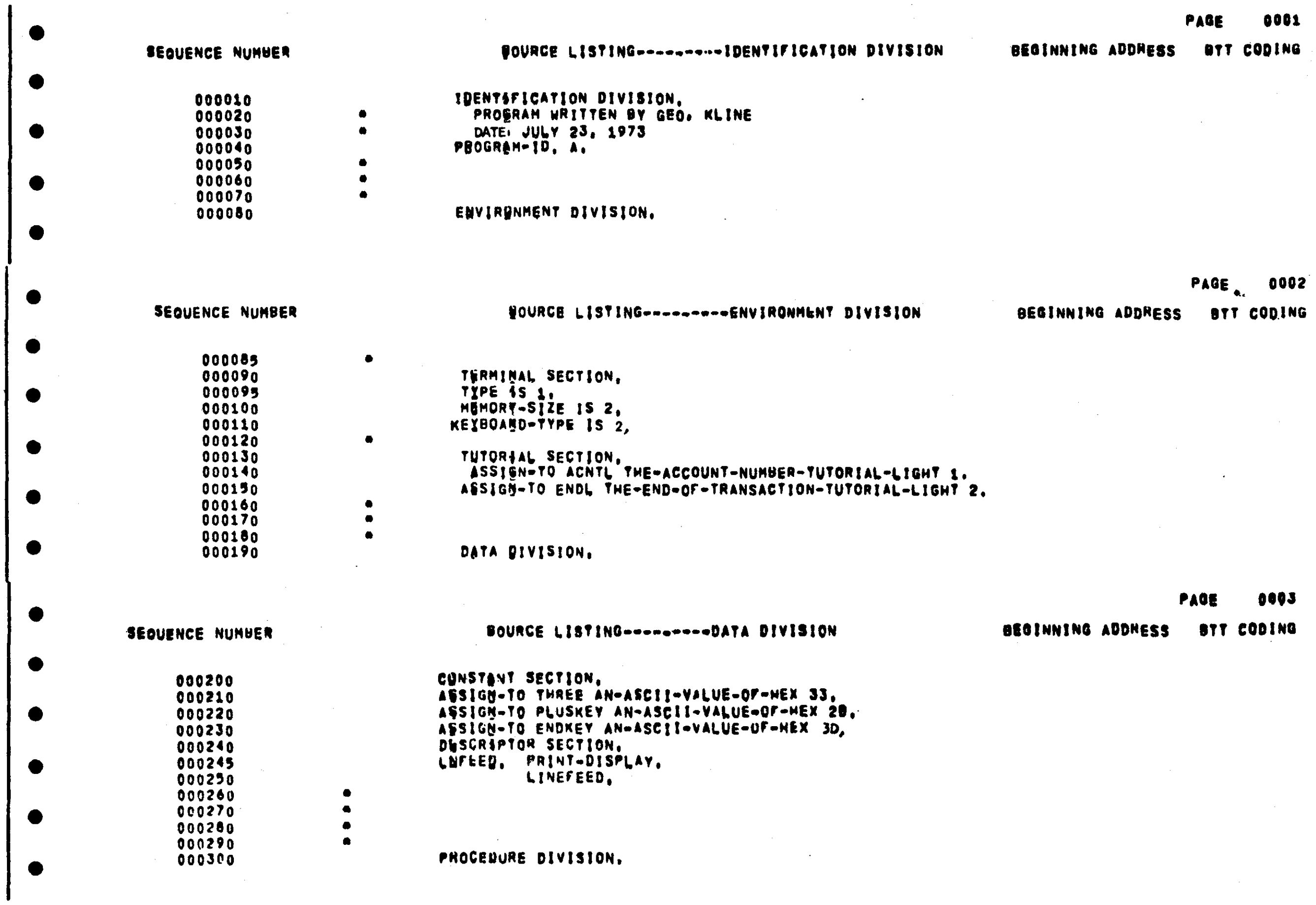
The central executive keeps the Terminal performing a number of activities in parallel by executing each of the above described programs in partitional segments.

CUSTOMER PROGRAMMING IN RAM

The customer programs the transaction sequences, tutorial lights, Journal and passbook printing formats, etc., through the use of program tables. In addition, a number of loadable BLU executive routines provided by Honeywell can also be loaded and called by the tables. These routines can be written for the user thus providing special function capability; e.g., a check digit routine on account number field may use the user's algorithm to validate the account number in a transaction. Since the functioning of the Terminal components has been programmed in firmware, the user is left with the programming that determines sequence and formats of transactions per his requirements. This helps reduce customer programming effort considerably, while not sacrificing flexibility.

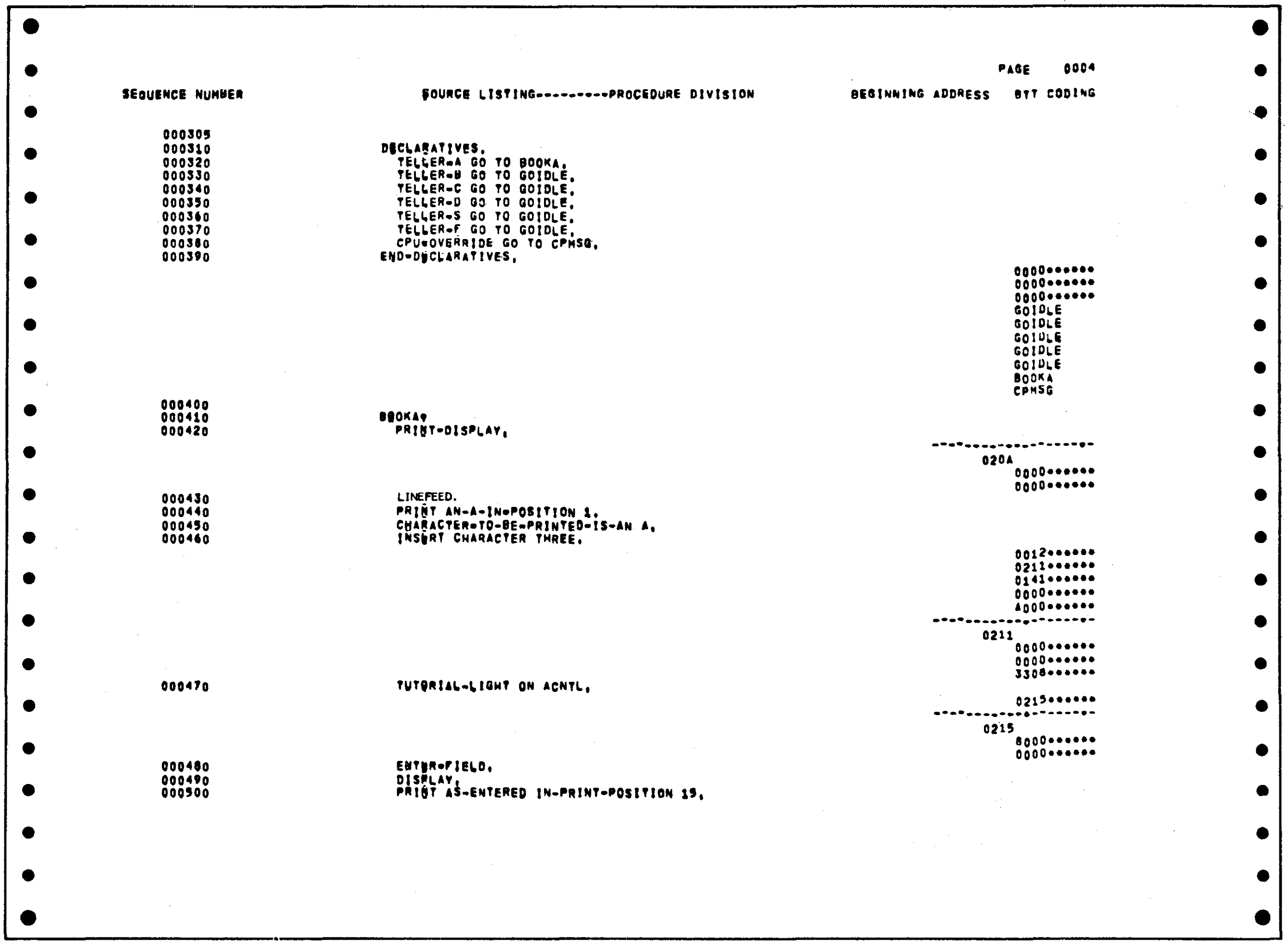
PROGRAMMING AID TO THE CUSTOMER (FITAL LANGUAGE)

In order to make user's programming and debugging effort still easier, Honeywell provides him with a COBOL based higher level application oriented language called "FITAL" and an associated "FITAL" compiler. "FITAL" compiler is written in Cobol and runs on Honeywell 2000 and 600 Series. The language is COBOL "like" such as the



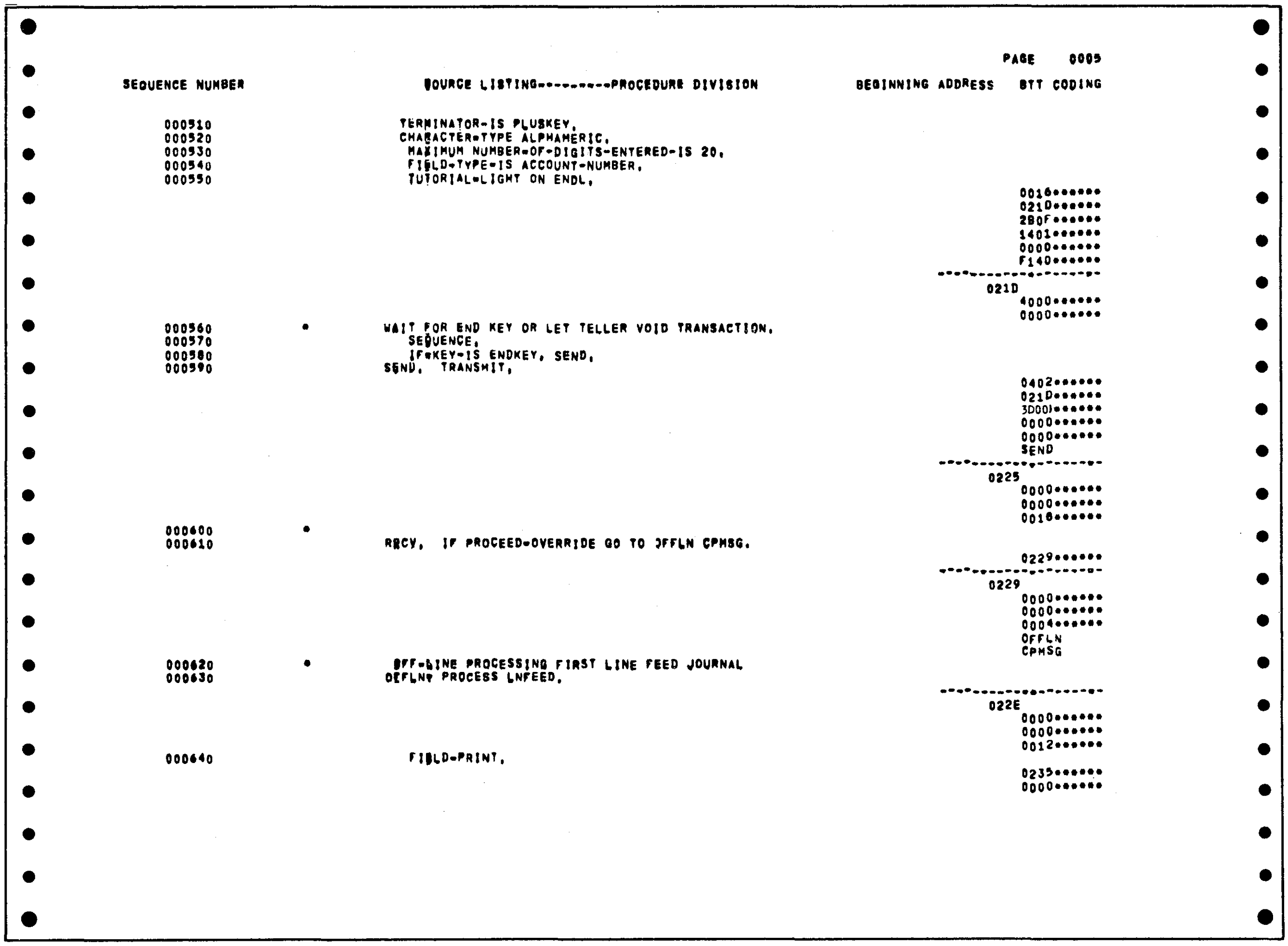
```
                                                                                          PAGE    0001
   SEQUENCE NUMBER                 SOURCE LISTING---------IDENTIFICATION DIVISION      BEGINNING ADDRESS   BTT CODING

      000010                       IDENTIFICATION DIVISION,
      000020          •              PROGRAM WRITTEN BY GEO. KLINE
      000030          •              DATE: JULY 23, 1973
      000040                       PROGRAM-ID, A,
      000050          •
      000060          •
      000070          •
      000080                       ENVIRONMENT DIVISION,

                                                                                          PAGE    0002
   SEQUENCE NUMBER                 SOURCE LISTING---------ENVIRONMENT DIVISION         BEGINNING ADDRESS   BTT CODING

      000085          •
      000090                        TERMINAL SECTION,
      000095                        TYPE IS 1,
      000100                        MEMORY-SIZE IS 2,
      000110                       KEYBOARD-TYPE IS 2,
      000120          •
      000130                        TUTORIAL SECTION,
      000140                         ASSIGN-TO ACNTL THE-ACCOUNT-NUMBER-TUTORIAL-LIGHT 1.
      000150                        ASSIGN-TO ENDL THE-END-OF-TRANSACTION-TUTORIAL-LIGHT 2.
      000160          •
      000170          •
      000180          •
      000190                        DATA DIVISION,

                                                                                          PAGE    0003
  SEQUENCE NUMBER                 SOURCE LISTING---------DATA DIVISION                BEGINNING ADDRESS   BTT CODING

     000200                       CONSTANT SECTION,
     000210                       ASSIGN-TO THREE AN-ASCII-VALUE-OF-HEX 33,
     000220                       ASSIGN-TO PLUSKEY AN-ASCII-VALUE-OF-HEX 2B,
     000230                       ASSIGN-TO ENDKEY AN-ASCII-VALUE-OF-HEX 3D,
     000240                       DESCRIPTOR SECTION,
     000245                       LNFEED,  PRINT-DISPLAY,
     000250                                LINEFEED,
     000260          •
     000270          •
     000280          •
     000290          •
     000300                       PROCEDURE DIVISION,
```

Figure 7—Sample program listing for FITAL



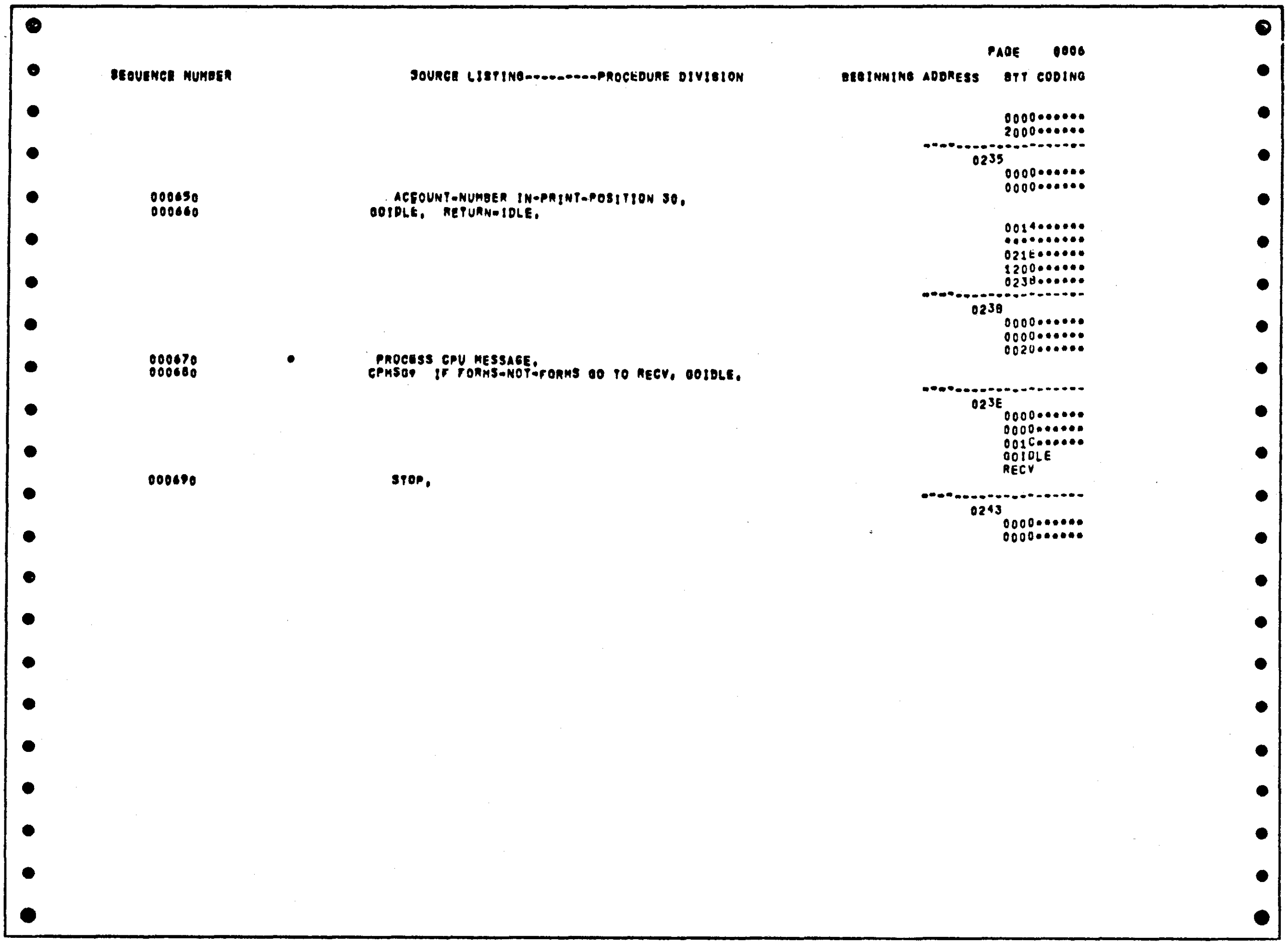
```
                                                                                          PAGE    0004
SEQUENCE NUMBER              SOURCE LISTING----------PROCEDURE DIVISION        BEGINNING ADDRESS   BTT CODING

    000305
    000310                   DECLARATIVES.
    000320                     TELLER-A GO TO BOOKA,
    000330                     TELLER-B GO TO GOIDLE,
    000340                     TELLER-C GO TO GOIDLE,
    000350                     TELLER-D GO TO GOIDLE,
    000360                     TELLER-S GO TO GOIDLE,
    000370                     TELLER-F GO TO GOIDLE,
    000380                     CPU-OVERRIDE GO TO CPMSG,
    000390                   END-DECLARATIVES,
                                                                                       0000••••••
                                                                                       0000••••••
                                                                                       0000••••••
                                                                                       GOIDLE
                                                                                       GOIDLE
                                                                                       GOIDLE
                                                                                       GOIDLE
                                                                                       GOIDLE
                                                                                       BOOKA
                                                                                       CPMSG
    000400
    000410                   BOOKA.
    000420                     PRINT-DISPLAY,
                                                                          ----------------------
                                                                                  020A
                                                                                       0000••••••
                                                                                       0000••••••
    000430                     LINEFEED.
    000440                     PRINT AN-A-IN-POSITION 1.
    000450                     CHARACTER-TO-BE-PRINTED-IS-AN A,
    000460                     INSERT CHARACTER THREE.
                                                                                       0012••••••
                                                                                       0211••••••
                                                                                       0141••••••
                                                                                       0000••••••
                                                                                       A000••••••
                                                                          ----------------------
                                                                                  0211
                                                                                       0000••••••
                                                                                       0000••••••
                                                                                       3308••••••
    000470                     TUTORIAL-LIGHT ON ACNTL,
                                                                                       0215••••••
                                                                          ----------------------
                                                                                  0215
                                                                                       8000••••••
                                                                                       0000••••••
    000480                     ENTER-FIELD,
    000490                     DISPLAY,
    000500                     PRINT AS-ENTERED IN-PRINT-POSITION 15,
```

Figure 7—Sample program listing for FITAL (Continued)



```
                                                                                                    PAGE    0005
SEQUENCE NUMBER              SOURCE LISTING----------PROCEDURE DIVISION          BEGINNING ADDRESS   BTT CODING

    000510                        TERMINATOR-IS PLUSKEY,
    000520                        CHARACTER-TYPE ALPHAMERIC,
    000530                         MAXIMUM NUMBER-OF-DIGITS-ENTERED-IS 20,
    000540                         FIELD-TYPE-IS ACCOUNT-NUMBER,
    000550                         TUTORIAL-LIGHT ON ENDL,
                                                                                                 0016*******
                                                                                                 021D*******
                                                                                                 2B0F*******
                                                                                                 1401*******
                                                                                                 0000*******
                                                                                                 F140*******
                                                                                   -----------------------
                                                                                            021D
                                                                                                 4000*******
                                                                                                 0000*******
    000560             *         WAIT FOR END KEY OR LET TELLER VOID TRANSACTION.
    000570                          SEQUENCE,
    000580                          IF-KEY-IS ENDKEY, SEND,
    000590                       SEND.  TRANSMIT,
                                                                                                 0402*******
                                                                                                 021D*******
                                                                                                 3D00)******
                                                                                                 0000*******
                                                                                                 0000*******
                                                                                                 SEND
                                                                                   -----------------------
                                                                                            0225
                                                                                                 0000*******
                                                                                                 0000*******
                                                                                                 0018*******
    000600             *
    000610                       RECY,  IF PROCEED-OVERRIDE GO TO OFFLN CPMSG.
                                                                                                 0229*******
                                                                                   -----------------------
                                                                                            0229
                                                                                                 0000*******
                                                                                                 0000*******
                                                                                                 0004*******
                                                                                                 OFFLN
                                                                                                 CPMSG
    000620             *          OFF-LINE PROCESSING FIRST LINE FEED JOURNAL
    000630                       OFFLN, PROCESS LNFEED.
                                                                                   -----------------------
                                                                                            022E
                                                                                                 0000*******
                                                                                                 0000*******
                                                                                                 0012*******
    000640                          FIELD-PRINT,
                                                                                                 0235*******
                                                                                                 0000*******
```

Figure 7—Sample program listing for FITAL (Continued)



```
                                                                                          PAGE    0006
  SEQUENCE NUMBER           SOURCE LISTING----------PROCEDURE DIVISION     BEGINNING ADDRESS   STT CODING

                                                                                          0000••••••
                                                                                          2000••••••
                                                                              ------------------
                                                                                      0235
                                                                                          0000••••••
     000650                   ACCOUNT-NUMBER IN-PRINT-POSITION 30,                        0000••••••
     000660                 GOIDLE,  RETURN-IDLE,
                                                                                          0014••••••
                                                                                          ••••••••••
                                                                                          021E••••••
                                                                                          1200••••••
                                                                                          023B••••••
                                                                              ------------------
                                                                                      023B
                                                                                          0000••••••
                                                                                          0000••••••
                                                                                          0020••••••
     000670       •          PROCESS CPU MESSAGE,
     000680                 CPMSG,  IF FORMS-NOT-FORMS GO TO RECV, GOIDLE.
                                                                              ------------------
                                                                                      023E
                                                                                          0000••••••
                                                                                          0000••••••
                                                                                          001C••••••
                                                                                          GOIDLE
     000690                   STOP,                                                       RECV
                                                                              ------------------
                                                                                      0243
                                                                                          0000••••••
                                                                                          0000••••••
```

Figure 7—Sample program listing for FITAL (Continued)

FUNCTION	HARD WIRED APPROACH	COMPLETELY PROGRAMMABLE TERMINAL APPROACH	BANK TELLER TERMINAL 7340 APPROACH
1. Flexibility in Customer Programming	Not Available	Available	Limited availability
2. Customer Programming Effort	None	Whole Terminal is to be programmed.	Programming kept to a minimum.
3. Systems Effort in Programming	None	Required in most cases.	None with the availability of FITAL.
4. Modification of Terminal Design	Very Expensive	Can be programmed.	Can be programmed using loadable executable routines.
5. Loading the Terminal at Power On Time	Not Required	Complete program has to be loaded in the memory.	Due to the firmware, the amount of program to be loaded is much shorter than that for the programmable Terminal.
6. Speed of Execution	Fast - No overhead of programming exists	Relatively slow as all the functions have to be programmed in software.	Not as fast as hard wired, but fast enough to maintain Terminal-Computer interface and Operator-Terminal interface without error or delay.

Figure 8—Advantages and disadvantages of Honeywell approach versus hard wired/completely programmable approach

example shown below:

FITAL program example

Consider a simple program to accept a transaction from Teller A only, print A to identify teller, insert an ASCII 3 into message to Central Processor (CP), accept an alpha/numeric field of data from the operation, wait for END OF TRANSACTION key from the operator, and then if:

- Off-Line—Print the field, then set Terminal idle
- On-Line—Send message to CP, receive and process response from CP, then set Terminal idle.

Operator transaction key stroke entry

Ⓐ (Alpha/Numeric Field) $\boxed{+}$ $\boxed{\text{END}}$

Journal Printing Layout:
(Printed at entry time from operator input)
A (Alpha/Numeric Field)
Position 1 Position 15
(Printed in response to a PROCEED command received from the CP via the communication line)
(Alpha/Numeric Field)
Linefeed Position 10

A transaction flow chart for the example above is shown in Figure 6. The program listing appears in Figure 7.

CONCLUSIONS

The Honeywell approach has provided a good compromise between a hard wired Terminal and a completely programmable Terminal. This has helped Honeywell provide a cost competitive Terminal with sufficient flexibility for the customer to tailor the Terminal to his needs. The rationale used in the design has been that the customer programming requirements should be kept to a minimum, restricted only to job oriented functions, while not sacrificing the flexibility to the customer. This was accomplished by fixing all real time and mechanism control functions in ROM while allowing user program control of all transaction related functions.

Figure 8 summarizes some of the advantages and disadvantages of the Honeywell approach in comparison with the hard wired or the completely programmable terminal approach. Honeywell feels that the savings in programming cost to the user and savings in hardware memory cost are sufficient to justify the minimal limitations in customer programmability.

ACKNOWLEDGMENT

The author wishes to acknowledge H. K. Barger, R. L. Bibles, M. Gilbert, G. H. Kline, A. F. Maggio, and G. B. Voelker for their valuable suggestions on the paper and Honeywell Information Systems for granting permission for presenting the paper.

Designing central processors with bipolar microcomputer components

by MARCIAN E. HOFF, JR.
Intel Corporation
Santa Clara, California

Most first and second generation MOS microcomputer products used single-chip fixed-control central processors or used specialized mask-programmed ROM's which made user microprocessing very cumbersome. Today, a third generation of microcomputer products using bipolar technology has appeared. These new components may be used to build controllers and computer central processing units (CPU's) in which the control structure is microprogrammed. By permitting microprograms in standard programmable ROM's, the bipolar microcomputer components offer much greater convenience for user microprogramming.

One such family of bipolar microcomputer components is the Intel 3000 series, a set of components realized with Schottky TTL technology. The two most important circuits in this family are the 3001 microprogram control unit (MCU) and the 3002 central processing element (CPE). The MCU determines the sequence of execution of micro-instructions from the control memory, and provides carry logic. The CPE represents a two-bit wide slice through the arithmetic, logic, register and data bus portions of a computer central processing unit. Several CPE's may be wired together to produce a central processing unit with arbitrary data bus width. For example, to produce a 16-bit wide data path, eight CPE's would be used.

Other members of the family include the 3003 fast carry chip, the 3212 input/output register chip, and the 3214 interrupt control chip. The control memory portion of a central processor or controller built with this family may be realized with standard field-programmable ROM's, mask-programmable ROM's or read/write memory (RAM's).

MICROPROGRAMMED CONTROL

The central processing unit of a general purpose computer contains: an arithmetic portion and a control portion. The basic operation of the control portion, passing through a sequence of states which select the next instruction from memory, and then execute a series of suboperations based on the instruction fetched, may be implemented via random logic or by the use of a table in a control memory. The latter technique is referred to as microprogrammed control.

The functions of the control portion of a microprogrammed-control central processing unit are very similar to the functions of a central processing unit. The terms "micro" and "macro" are used to distinguish the operations of the control unit from those of the realized central processor. The central processor, under the direction of micro-instructions read from its control memory, fetches macro-instructions from main memory. Each macro-instruction is then executed as a series of micro-instructions. The main memory contains a macroprogram, while the central processor is defined by the microprogram contained in the control memory.

Thus, within a microprogrammed machine, there are at least two levels of control and two levels of programming to be considered. The designer of the central processor is usually concerned with the definition of the macro-instruction set and its realization as a microprogram. The final user of the central processing unit seldom needs to be aware that the CPU was realized using microprogramming. A description of the macro-instruction set is usually sufficient for his purposes.

The microprogrammed approach is useful for bipolar microcomputers because complex macro-instruction sets can be realized as sequences of relatively primitive micro-instructions. The logic of the final macromachine remains relatively simple, with most of the complexity being represented by the contents of the control memory.

When using the Intel bipolar microcomputer family, the 3001 MCU implements most of the functions of a microprogram control unit. When used with the 3002 CPE slice, the basic micro-instruction functions are established, although additional logical elements drawn from standard TTL families may be added which will alter or enhance the micro-instruction set.

DESIGNING CENTRAL PROCESSING UNITS

The steps in the development of a microprogrammed central processor design are:

1. Selection of the macro-instruction set
2. Hardware design
3. Writing and checkout of the microprograms

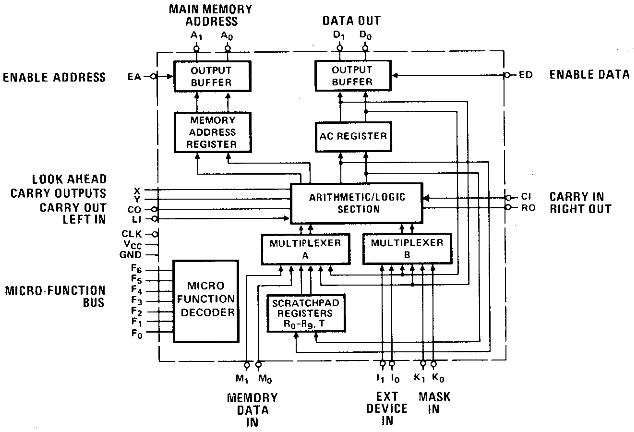


Figure 1—3002 CPE block diagram

The most efficient designs will result from considering all three steps together to insure that the macro-instruction set formats are compatible with the microcomputer family members, and that operation-code formats result in the simplest microprogram flow.

The macro-instruction set may be register or stack oriented, and may be original or a copy of another machine. In general, lower cost and higher performance will result when an original macro-instruction set is developed to use the microcomputer family features most effectively. In most cases the three special purpose registers and a few of the general purpose CPE registers must be reserved for microprogram "bookkeeping" operations. As a result, when realizing macro-instruction sets which use a large number of registers, an external register file will have to be added. If more than eight bits of operation code are used in a macro-instruction, the interpretation of the macro-instruction becomes more complex. In some cases, additional logic may have to be added to a basic hardware design.

HARDWARE DESIGN

A typical CPU built using the Intel bipolar microcomputer set will consist of an array of CPE chips, one MCU, and a control memory. The array of CPE chips realizes the arithmetic and logical functions and registers of the CPU, while the combination of MCU and control memory realizes the control portion.

The CPE array realizes ten general purpose (R0-R9) and three special purpose registers (MAR, AC, and T), each with a width equal to the array width (i.e., 16 bits wide for eight CPE slices). The array can be wired for ripple carry operation or may use the 3003 look-ahead carry generation.

The CPE array has six buses for communication with external circuitry. (Figure 1 shows a block diagram of the CPE slice.) Four of the buses are used primarily to communicate with memory and I/O devices while those remaining, the function control bus and the control memory data bus, enable the control portion of the processor to drive the CPE array. The seven-bit wide function control bus is driven by outputs from the control memory to force the CPE array to execute the desired operation. The control memory data bus (also referred to as the K-bus) allows the control memory to supply constants and masks to the arithmetic array.

In effect, the K-bus increases the effective number of microfunctions executable by the CPE array. For example, data loaded into the CPE array from memory may be masked by the K-bus. In other operations, involving the AC and a general purpose register, the K-bus can mask off the AC register. Thus if the K-bus is all zeros, the AC does not contribute to the result. The K-bus may also be added to any register.

Although the K-bus is potentially the same width as the other data buses, the number of masks and constants used in a typical CPU is usually small enough that fewer bits are needed. Often several K-bus inputs can be connected together and driven by a single control memory output line.

The actual microfunctions implemented by the CPE are listed in Table I. The microfunction standard mnemonics for both K-bus equal to all zeros and K-bus equal to all ones are also shown in Table I. For other values on the K-bus, the mnemonic for an all-one K-bus is used.

The microprogram control unit establishes the micro-instruction execution sequence as a function of three data sources: a seven-bit wide field in the control memory, carry logic within the MCU which communicates with the carry circuits of the CPE array, and macro-instruction operation codes. Figure 2 shows a block diagram of the MCU chip. To permit the microprogram control unit to determine the operation code portion of the macro-instruction being executed, an eight-bit wide path is provided from the memory data bus to the microprogram control unit chip. This path allows the microprogram control unit to examine eight bits of the operation code portion of the

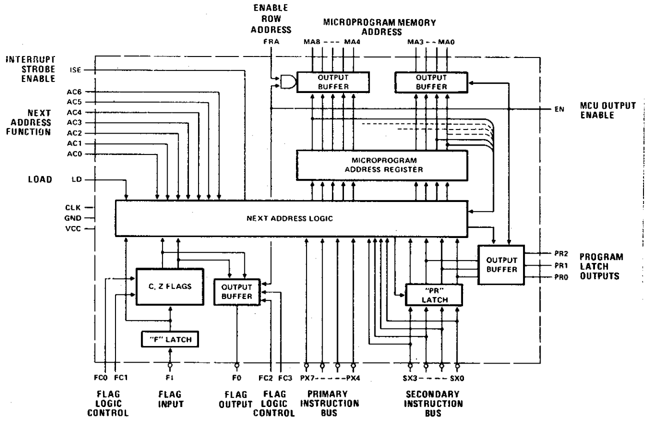


Figure 2—3001 MCU block diagram

TABLE I--3002 CPE Microfunctions

Function Bus $f_6f_5f_4f_3f_2f_1f_0$	Logic Function	Mnemonic (K-Bus = 1)	Mnemonic (K-Bus = 0)
000XXXX[2]	R+(AC∧K)+CI→R,AC	ALR	ILR
000101X[2]	M+(AC∧K)+CI→AT	AMA	ACM
000111X[2]	Shift Right AT for K=0	-	SRA
001XXXX[1]	K∨R→MAR,R+K+CI→R	DSM	LMI
001101X[2]	K∨M→MAR,M+K+CI→AT	LDM	LMM
001111X[2]	$(\overline{AT} \vee K)+(AT \wedge K)+CI \rightarrow AT$	DCA	CIA
010XXXX[1]	(AC∧K)-1+CI→R	SDR	CSR
010101X[2]	(AC∧K)-1+CI→AT	SDA[3]	CSA
010111X[2]	(I∧K)-1+CI→AT	LDI	-
011XXXX[1]	R+(AC∧K)+CI→R	ADR	INR
011101X[2]	M+(AC∧K)+CI→AT	-	-
011111X[2]	AT+(I∧K)+CI→AT	AIA	INA
100XXXX[1]	R∧(AC∧K)→R	ANR	CLR
100101X[2]	M∧(AC∧K)→AT	ANM	CLA[3]
100111X[2]	R∧(I∧K)→AT	ANI	-
101XXXX[1]	K∧R→R	TZR	-
101101X[2]	K∧M→AT	LTM	-
101111X[2]	K∧AT→AT	TZA	-
110XXXX[1]	R∨(AC∧K)→R	ORR	NOP
110101X[2]	M∨(AC∧K)→AT	ORM	LMF
110111X[2]	AT∨(I∧K)→AT	ORI	-
111XXXX[1]	$R\overline{\oplus}(AC \wedge K) \rightarrow R$	XNR	CMR
111101X[2]	$M\overline{\oplus}(AC \wedge K) \rightarrow AT$	XNM	LCM
111111X[2]	$AT\overline{\oplus}(I \wedge K) \rightarrow AT$	XNI	CMA

1. XXXX = 0000 to 1001 to select R = RØ to R9, 1100 for R = T, 1101 for R = AC.
2. X = 0 for AT = AC, X = 1 for AT = T.
3. SDA and CLA are the same as SDR and SDA respectively, except only AC or T may be used.

TABLE II—MCU Jump Microfunctions

Mnemonic	Description	Function							Next Row					Next Column			
		AC_6	5	4	3	2	1	0	MA_8	7	6	5	4	MA_3	2	1	0
JCC	Jump in current column	0	0	d_4	d_3	d_2	d_1	d_0	d_4	d_3	d_2	d_1	d_0	m_3	m_2	m_1	m_0
JZR	Jump to zero row	0	1	0	d_3	d_2	d_1	d_0	0	0	0	0	0	d_3	d_2	d_1	d_0
JCR	Jump to current row	0	1	1	d_3	d_2	d_1	d_0	m_8	m_7	m_6	m_5	m_4	d_3	d_2	d_1	d_0
JCE	Jump in column/enable	1	1	1	0	d_2	d_1	d_0	m_8	m_7	d_2	d_1	d_0	m_3	m_2	m_1	m_0
JFL	Jump/test F-latch	1	0	0	d_3	d_2	d_1	d_0	m_8	d_3	d_2	d_1	d_0	m_3	0	1	f
JCF	Jump/test C-flag	1	0	1	0	d_2	d_1	d_0	m_8	m_7	d_2	d_1	d_0	m_3	0	1	c
JZF	Jump/test Z-flag	1	0	1	1	d_2	d_1	d_0	m_8	m_7	d_2	d_1	d_0	m_3	0	1	z
JPR	Jump/test PR-latches	1	1	0	0	d_2	d_1	d_0	m_8	m_7	d_2	d_1	d_0	p_3	p_2	p_1	p_0
JLL	Jump/test left PR bits	1	1	0	1	d_2	d_1	d_0	m_8	m_7	d_2	d_1	d_0	0	1	p_3	p_2
JRL	Jump/test right PR bits	1	1	1	1	1	d_1	d_0	m_8	m_7	1	d_1	d_0	1	1	p_1	p_0
JPX	Jump/test PX-bus	1	1	1	1	0	d_1	d_0	m_8	m_7	m_6	d_1	d_0	x_7	x_6	x_5	x_4

Symbol	Meaning
d_n	Data on address control line n
m_n	Data in microprogram address register bit n
p_n	Data in PR-latch bit n
x_n	Data on PX-bus line n (active LOW)
f, c, z	Contents of F-latch, C-flag, or Z-flag, respectively

macrolevel instruction at the same time that the instruction is passed to the arithmetic array.

Because only eight bits of operation code information can be passed directly to the MCU, the set is best adapted to instruction sets in which all of the macro-operation code information is confined to 8 bits. However, other macroinstruction sets can be realized by saving any remaining bits of the operation code in the CPE array or in an external register. The saved bits are tested later by routing them to the MCU, either through the 8 line macro-instruction port or via the carry logic associated with the MCU-CPE combination.

The MCU characteristics provide a jump operation, either conditioned or unconditioned, in every microinstruction. Each of the jump operations has a restricted range of destinations, so that the placement of the microinstruction in the control memory determines which of the other micro-instructions may precede or follow it.

The nature of the different jumps provided by the MCU is made most visible by viewing the microprogram memory as if the words were organized as 16 columns by 32 rows. (The MCU directly supports 512 words of control memory, although larger or smaller control memories may be used.) The unconditional jumps provide for vertical

TABLE III—MCU Carry Logic Functions

Type	Mnemonic	Description	FC_1	FC_0
Flag Input	SCZ	Set C-flag and Z-flag to f	0	0
	STZ	Set Z-flag to f	0	1
	STC	Set C-flag to f	1	0
	HCZ	Hold C-flag and Z-flag	1	1
Type	Mnemonic	Description	FC_3	FC_2
Flag Output	FFO	Force FO to 0	0	0
	FFC	Force FO to C-flag	0	1
	FFZ	Force FO to Z-flag	1	0
	FF1	Force FO to 1	1	1

movement in a column, horizontal movement within a row, or a jump to any location in the first row, known as row zero. All of these jumps are achieved by appropriate mapping of microfunction control bits and previous microprogram address bits into the next microprogram address state. Table II lists both the unconditional and conditional jumps provided by the MCU. Note that the JPX operation tests four bits of the PX-bus (macro-operation input bus) while loading the remaining four bits into the PR-latch within the MCU chip. The JPR, JRL and JLL microfunctions test the contents of the PR-latch without altering them, while the JCE microfunction executes an unconditional jump while enabling three of the four stored bits to output pins on the MCU, where the signals may be used to override CPE register selection signals from the control memory.

The JFL, JCF and JZF microfunctions test the input carry signal or one of two flip-flops within the MCU which are associated with the array logic of the MCU. Either or both of these flip-flops (designated the C and Z flags) may be set to the carry input signal and the carry output signal may be set to logic 0, logic 1 or to the logical value loaded into C or Z. Table III lists the flag control functions. The symbol *f* represents the carry input signal, temporarily held in the F latch of the MCU for testing.

A "pipelined" mode of operation may be implemented by placing a register of D flip-flops between the control memory outputs and the circuitry controlled by those outputs. This register causes the execution of each microinstruction to overlap the fetching of the next microinstruction. The seven control lines which issue microinstruction sequence information to the MCU are not routed through D register when the pipelined mode is used.

Figure 3 shows a block diagram illustrating the organization of a central processing unit using the set. The block diagram shows the basic modules of a pipelined CPU: the CPE array, the MCU, the control or microprogram memory, and the pipeline register. Four of the data busses associated with the CPE array are shown: (1) The address bus to memory; (2) The data bus to memory; (3) The data bus from memory, with its path for operation code data to the MCU; (4) The constant bus from the pipeline register.

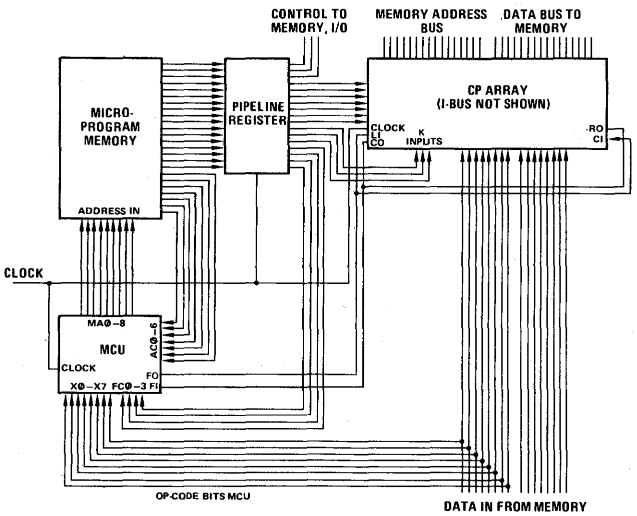


Figure 3—Pipelined central processing unit block diagram

In addition, the carry logic bus from the pipeline register to the MCU and the micro-instruction sequence logic from the control memory to the MCU are shown. Signals from additional control fields to such external logic as memory and I/O control are shown as an output bus from the pipeline register, although some such signals may come directly from the control memory.

THE WRITING OF THE MICROPROGRAMS

Once the macro-instruction set has been chosen, and the hardware design established, the designer must proceed to write the microprograms for the system. To simplify the writing of these microprograms, a standardized micro-assembly language will be used, in which symbolic representations of the various control functions are used. The standard mnemonics of CPE and MCU microfunctions have been included in Tables I, II and III.

Programs written in the micro-assembly language have two main parts—a declaration part in which various aspects of the control word, etc. are defined and a specification part in which the contents of each word are symbolically declared. Provision is made for comment statements throughout the program so that the programmer may explain the functions being performed.

The main body of the program, the specification part, defines the sequences of states to be executed, and the operations which take place for each state. The main effort in writing a microprogram will be extended in developing this section.

Each statement of the specification part of the program defines the action (and location) of one micro-instruction, i.e., one work of control memory. The statement will declare, either directly or by default, the contents of each control field for the specified micro-instruction. Furthermore, the statement will include assignment information designating the address in control memory where the statement is located.

A specification statement consists of one or more labels followed by a series of control field specifications. A colon after an entry indicates that it is a label. The contents of the control fields are indicated symbolically, using either standard symbols or user-defined symbols, or by an equation of the type

$$\mathrm{FNM} = 101\mathrm{B}$$

where FNM is a name associated with the field. The entry 101B implies the binary value 101.

Each symbol is associated with only one field, so that the various symbols can be uniquely interpreted by the assembler. A number of symbols are predefined for the assembler, and are not to be used except as provided for the MCU and CPE functions, and a number of directives to the assembler.

The current versions of the micro-assembler do not do this placement. However, the placement procedure, which is something similar in complexity to wiring a printed circuit board, is easily done after the microprogram is written. A flow chart showing the desired micro-program sequences is used to determine where clusters of conditional jumps are located. The most complex clusters are placed first, then simpler clusters, and finally the unconditional sequences.

To realize a central processing unit, the microprogram must not only supervise the fetching of macro-instructions and implement the appropriate sequences for their interpretation and execution, but must also provide for such additional functions as initialization and interrupt handling. Initialization, done when power is first applied to the central processor, is achieved by executing a series of micro-instructions which clear and initialize various machine registers. The "load" pin of the MCU can be used to input a starting address from the PX-bus to the micro-program address register. If the PX-bus is held high when power is first applied and the load input is activated, the MCU will start in control memory address zero.

Interrupts are usually accepted after completion of a micro-instruction, just before fetching the next macro-instruction. To provide interrupt handling, a sequence of interrupt service micro-instructions would be executed rather than fetch the next macro-instruction. Upon completion of the interrupt service, which usually involves at least saving the macro-program counter and loading a new value, the fetch sequence is entered, which results in fetching the first instruction of the interrupt service macro-program. The MCU provides for interrupt if the first instruction of the macro-instruction fetch sequence is located in control memory address 15 (i.e., in row zero, column 15). The last micro-instruction of each macro-instruction execution sequence then contains the microfunction JZR to row zero, column 15. When this microfunction is executed, the MCU produces an interrupt strobe signal which may be used by external interrupt logic to disable the MCU row address to the control memory. An alternate row address can then be forced to the control memory (usually row 31, all row address leads high) so that a different micro-instruction than the first fetch micro-instruction is executed, i.e., the first interrupt micro-instruction is executed. This first interrupt micro-instruction should contain a JZR or JCC microfunction as its first jump. Once in the interrupt sequence, the normal MCU control is re-enabled, and execution proceeds until a jump to the macro-instruction fetch sequence is executed. There are many interrupt handling options available through microprogram sequences, hardware connections, and the type of jump used to enter the fetch routine.

Microprograms to realize a given micro-instruction set will differ for pipelined and non-pipelined machines. The major differences are associated with those conditional jumps in the microprogram which test the results of arithmetic or logical operations of the CPE array. In a pipelined machine, these results are delayed by one micro-instruction, so that the conditional jump must be delayed by at least one micro-instruction.

In some cases, the pipeline will result in wasted states (NOP's), but most pipelined microprograms will have very

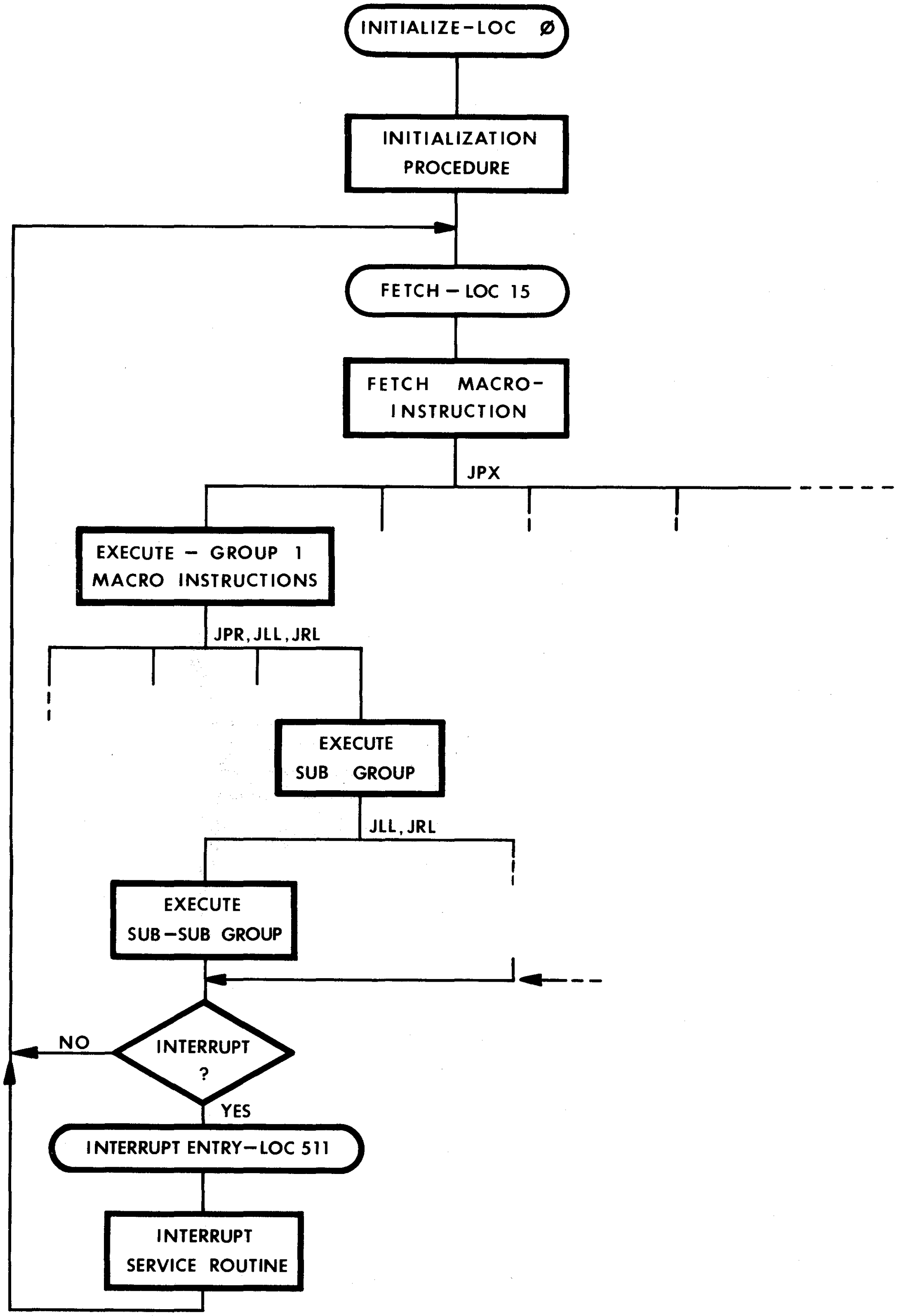


Figure 4—Basic central processing unit microprogram flow chart

few such wasted locations. In most cases, the otherwise wasted location can be used to set up for the next conditional jump or to do some initial processing for the next execution sequence.

A typical statement of the specification section of a microprogram might take the form:

123: LAB: ILR(R3) FFO STZ JFL(MC TC);

The number 123 followed by a colon tells the assembler that the micro-instruction is assigned to location 123 in control memory. (Binary for 123 is 001111011—placing the word in row 7, column 11 when the control memory is treated as an array of 32 rows and 16 columns.) The symbolic label LAB (the colon indicates a label) is also associated with this location. ILR(R3) indicates that the contents of register 3 are to be conditionally incremented and copied to the AC register, while FFO forces the carry input to a logic zero, so that the incremented operation does not take place. STZ indicates that the Z flip-flop of the MCU is to be set by the results, so that, as no carry can result, the Z flip-flop will be set to a logic zero. These symbols are standard symbols, with ILR associated with the CPE and FFO and STZ associated with the MCU carry logic. The JFL tests the carry output line for a conditional jump to either the statement labeled NC or to the statement labeled TC. It is also a standard symbol. Note that, if the machine is pipelined, this conditional jump tests the results of the previous instruction, not of this one. The semicolon indicates the end of the statement.

In the statement above, no information was provided for the K-bus. The assembler will provide the appropriate value associated with the ILR operation, i.e., the K-bus at all zeros.

The first or declaration part of a microprogram defines all of the user symbols except the labels, which are defined in the specification part of the program. The user symbols may include redefined or renamed combinations of other symbols, or may attach names to various control fields or to states within a control field.

The flow diagram for a typical central processor will follow the form shown in Figure 4.

In general, the best macro-operation sets will be organized so that the bits tested at the first conditional jump (the JPX) determine the initial steps in the instruction processing, for example, the first step will usually involve address calculation. Thus the best macro-operation sets will be encoded to allow addressing mode to be tested first, with the detailed use of the address or data from the address left for later testing in the microprogram.

The set has been used to realize a variety of machines including a stack-architecture 16-bit CPU oriented toward high-level languages, a 7-register 16-bit CPU which used base registers to provide full code and data relocatability, a multi-register 16-bit CPU which used a section of fast main memory to implement 256 registers, and a disk controller for a removable cartridge flying-head disk. Although the machines constructed so far have all been 16-bit machines, both 12-bit and 32-bit machines have also been considered. The typical CPU constructed used an average of 3-5 micro-instructions per main memory cycle. With a pipelined machine cycle time of 150nsec, a main memory of 300nsec access time results in an average of about one microsecond per memory cycle.

Bipolar microprocessor design configurations

by DAVID C. WYLAND
Monolithic Memories, Inc.
Sunnyvale, California

INTRODUCTION

Implementation of microprocessors in bipolar technologies promises performance equal to or exceeding that of state-of-the-art minicomputers when measured in terms of speed and instruction set power. The design approaches for bipolar microprocessors are similar to those for MOS types, with some significant differences in emphasis.

The term "microprocessor" was originally applied to some of the first single chip MOS microprocessors, such as the Intel 4004 and 8008 units. Shortly after the introduction of these chips, National Semiconductor introduced their General Purpose Control Processor (GPCP) chip set, now known as the IMP-16. The use of the word microprocessor was then broadened to include sets of chips which would implement the Central Processing Unit (CPU) function of a computer. To date, the use of the term has been further diluted to include almost any combination of less than 20-40 integrated circuits which will implement the CPU function.

The simple definition of the word "microprocessor" is "very small processor". This defines a unit which meets a minimum functional definition of the processing and control portion of a stored program computer, and which is "very small" (i.e., "micro"). The utility of the microprocessor lies in its small size: a card containing 25 integrated circuits can do the work of a 300 integrated circuit minicomputer in many cases. This reduction in size provides a corresponding reduction in system cost and power requirements.

MICROPROCESSOR DESIGN CONFIGURATIONS

Microprocessor designs can be grouped into three categories:

1. Single chip
2. Multiple chip set, custom design chips
3. Multiple chip set, general purpose design chips

The single chip microprocessor is a fixed design, fixed instruction set processor incorporated onto one integrated circuit chip. This has the potential advantage of lowest microprocessor system cost but the disadvantages of a fixed, lower performance instruction set; low to medium speed, and limited word size due to limitations of chip size and power dissipation. Most current MOS microprocessors are in this category. No bipolar microprocessors are currently available in this category due to the high level of integration required.

Multiple custom chip microprocessors consist of a set of chips specifically designed to work together to perform the microprocessor function. The National Semiconductor IMP-16, etc., microprocessor series is based on such a set of custom MOS chips. The advantages of this approach are:

1. Segregation of system functions into chips which are potentially small and efficient to integrate.
2. A powerful, sophisticated instruction set can be implemented because of the availability of more effective chip area with multiple chips.
3. Higher speed potential due to higher power dissipation capability, allowing better speed/power tradeoff.

The disadvantages are:

1. Higher system cost due to the requirement of several chips. This must be weighed against performance not available in a single chip approach.
2. Relatively fixed instruction set capability. This is because instruction decoding can be integrated efficiently only as a fixed decoder for a fixed instruction set. Also, some secondary items such as carry and shift controls can be handled efficiently only if their configuration is fixed. Most bipolar microprocessors are of the custom chip set configuration. Examples include the Intel, Raytheon, and Fairchild bipolar chip sets, as well as the TI I^2L set. These chip sets consist of data flow chips in 2 or 4 bit slices which are assembled to form 16 bit, etc., machines and one or more control chips, and in some cases some external microprograms ROMs.

The third microprocessor design configuration is the multiple general purpose chip set. In this approach, a combination of standard data flow slice chips, ROMs, and some TTL is used to implement the microprocessor function. The advantages of this approach include those of the above approach: efficient integration of smaller chips, powerful instruction set potential, and high speed potential. However, three additional advantages result:

1. Flexibility in system design, allowing the same hardware components to be used efficiently for a variety of tasks including I/O controllers, central processors, and special purpose peripheral processors.
2. Lower potential costs by using standard, high volume components designed for a broad market rather than custom, lower volume components designed for a specific market.
3. Higher second source potential, since high volume general purpose components are more likely to be second-sourced than custom types.

The disadvantage is that general purpose multi-chip designs have a higher chip count than custom multi-chip designs, in some cases. For example, a 16 bit bipolar microprocessor may have a total of:

1. Five chips in a maximally integrated custom chip set configuration, consisting of a control chip and four, four bit data flow slices.
2. Twenty chips in a more modular custom chip set configuration, consisting of eight, two bit data flow chips, eight microprogram memory chips, and four control chips.
3. Sixteen chips in a general purpose component design consisting of four, four bit data flow chips, three microprogram ROMs, one instruction decode ROM, and eight TTL chips for control.

MULTIPLE CHIP CONFIGURATION COMPONENTS

Multiple chip microprocessor designs typically separate system functions into three chip types:

1. Data flow chips which contain the registers, multiplexers, and arithmetic portions of the system.
2. Control chips which provide the instruction fetch and execution sequencing as well as miscellaneous status and test functions, such as carry control and shift gating.
3. Microprogram memory chips (optional, used only in conjunction with control chips based on microprogram designs). These are typically standard ROM/P.ROM chips assembled to form a memory unit.

The data flow chips are usually designed as 2 or 4 bit modules which can be assembled into 8, 12, 16, 24, etc., bit word widths as required. This accommodates a variety of system word sizes as well as reducing the size of the individual chip to allow higher speed and more powerful architectures. The data flow chips provide the major reduction in processor chip count. This is because each four bit slice may replace 15-25 MSI components, such as multiplexers, register files, and arithmetic logic units. Thus, four data flow chips are equivalent to 60-100 MSI chips. In the case of bipolar data flow slices, the comparison with MSI is fairly direct.

Microprogrammed bipolar microprocessor designs typically use standard ROM and P.ROM memory chips. This is because there is a broad line of high performance bipolar ROMs and P.ROMs on the market, available in high volume production from a number of sources. This is significantly different than the case with MOS. The MOS ROMs on the market are much slower, 2.0 microseconds versus 50 nanoseconds for bipolar, much more limited in selection of ROMs and P.ROMs, and are not as standardized in power supply and interface voltage specifications as a group when compared to bipolar ROMs and P.ROMs.

The control section of a multi-chip microprocessor provides the sequencing logic to fetch, decode, and execute all processor instructions. This sequencing logic controls the data flow chips and memory either directly or through a microprogram memory, depending on the design. The control section can be one or several chips depending upon whether a state logic or a microprogrammed design is used, respectively. These two control design approaches have quite different implications.

CONTROL FLOW INTEGRATION

A minimum chip count 16 bit multiple custom chip bipolar microprocessor could be designed using one chip for control and four, four bit data flow chips. The one control chip in this case would be of a state logic, "hardwired" design where the data flow and system control lines are driven by a complex combination of flip-flops and gates. This would result in minimum chip size and maximum speed at the cost of a fixed instruction set. Although efficient, this design would require extensive development and modelling plus market research to insure that the chosen instruction set is both saleable over the long term and capable of being implemented. The basic trade-off is therefore reduction in total chip count from approximately 20 to 5 at the expense of a long development cycle and a market restricted to a fixed instruction set.

The other design approach is based on microprogram design. In this case, the control chip drives the system and data flow control lines through a microprogram ROM external to the chip. Although it is possible to design a single control chip using microprogram techniques, as was done in the case of the National Semiconductor IMP series, the existence of high performance, low cost, standard bipolar ROM chips makes the multiple chip configuration more cost effective.

In a multi-chip, microprogram control design, the control chip generates a sequence of ROM addresses and provides some secondary functions including carry controls and shift gating. The major chip design elements are an instruction register, a ROM address register/counter, and ROM next address logic which decodes the contents of the instruction register and system status indicators to generate the next ROM state. The key to control chip design is in the ROM next address decoder.

The designer of a microprogram control chip faces a dilemma. The instruction decode logic can be designed along one of three lines:

1. For a fixed, unique instruction set, which will result in minimum logic and chip area.
2. For broad general purpose use with a wide variety of potential instruction sets.
3. For the compromise case: a relatively fixed instruction set with capability for some expansion and modification.

The problem is that none of these three approaches works effectively for different reasons.

If a fixed instruction set is defined, the argument for custom, random logic control chips as described earlier becomes strong. If there is some control over the instruction set and instruction formats, the design difficulty can be greatly reduced. This will result in the most efficient design at the cost of development time.

Alternatively, it is possible to make powerful fixed machines with simple microprogram control architectures using standard ROMs. For instance, the microprogram control logic for a multiple register machine similar to machines of the Interdata, IBM 360/370, and Modular Computer class can consist of an instruction register, a 256×8 ROM (one chip), an 8 bit synchronous counter (2-74163s), and two TTL chips to test system status conditions. When used in conjunction with a microprogram ROM and 4 bit slices, such a Monolithic Memories' 6701s, a 16 bit machine with 256 possible instructions can be built.

The fixed decode microprogram control chip has the problem that it does not provide the advantages of either the custom logic chip design (minimum system part count) or the standard ROM assembly design (low chip count, standard components, and the ability to decode a large instruction set in a flexible fashion). The system definition provided by the fixed decode microprogram control chip is therefore that of restricting the design to a particular instruction set without providing a corresponding improvement in packaging efficiency or instruction power over what could be obtained with a very few ROM and TTL parts.

Designing a microprogram control chip for broad, general purpose use is also difficult: the decoder design has no shape. A rather general approach could be implemented by using a large programmable logic array (PLA) to decode the instruction register and provide inputs to a ROM address counter/register. Another aspect of the problem is that good general purpose designs are usually the result of evolution through a series of specific designs followed by a consolidation of good features. Since the component count for a typical discrete ROM decoder is typically low, there is a good argument for a standard logic chip approach to the general purpose control logic as well.

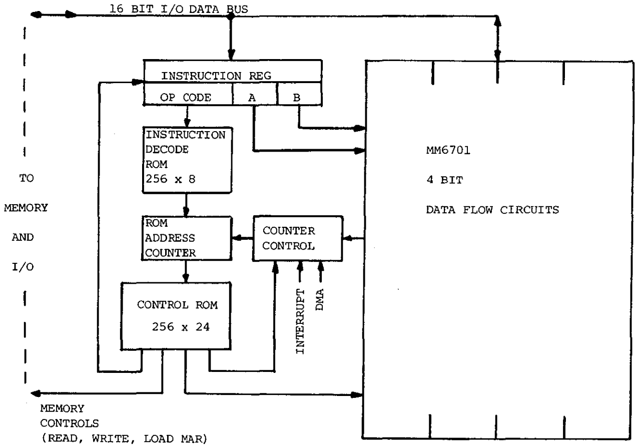


Figure 1—CPU block diagram

This leaves us with the compromise position: a partially structured instruction set for a definable control chip with some generality and modification capability. Since a microprogram control chip does not appear to offer an advantage for the fixed instruction set case, nor for the general case, it is difficult to derive an advantage for the intermediate, semi-fixed case.

CONTROL FLOW INTEGRATION: BIPOLAR VS. MOS

One may ask why there is little advantage to a dedicated microprogram control chip in a multi-chip bipolar microprocessor configuration. The reason is the existence of broad lines of high performance, low cost TTL ROMs and P.ROMs; and the broad line of TTL logic in general.

MOS ROMs and P.ROMs are not available in the same variety as TTL types and their performance is much lower. MOS logic level interfaces and power supply requirements have not been as standardized as TTL, and there is no standard set of universally compatible, general purpose MOS support logic as there is with TTL. This makes the design of an MOS multi-chip microprocessor more difficult. Because of the lack of both high speed MOS microprogram ROMs with standard logic level interfaces and a standard MOS support logic, the control chip must drive the system control lines directly. Either a state logic design or a tightly packed microprogram design with all microprogram ROM on-chip must be used.

With a broad availability of bipolar ROMs, P.ROMs, and TTL support logic, the necessity and in fact justification for a dedicated control chip is greatly reduced. Except for the case of a one chip controller for a dedicated instruction set, it would appear that power, cost effectiveness, and flexibility favor a combination of data flow chips with standard ROMs and some TTL for multi-chip bipolar microprocessors.

CONTROL FLOW WITH STANDARD COMPONENTS: AN EXAMPLE

Figure 1 shows a block diagram of a 16 bit bipolar microprocessor using Monolithic Memories' 4 bit data flow chips with ROMs and TTL logic for the control section. For a total of less than 25 chips, one can construct a 16 register machine with interrupt and Direct Memory access capability, indexed indirect addressing, hardware multiply and divide, double precision (32 bit) arithmetic capability, and stack oriented instructions. Performance of 900 nanoseconds for a register to register ADD instruction, 1.2 microseconds for a register load instruction, is typical.

SUMMARY

This paper has explored the different design configurations for bipolar microprocessors. Three distinct categories result:

1. The single chip microprocessor which has the lowest system cost with moderate performance.
2. The multiple chip design with a dedicated instruction set and dedicated control chip design, which results in high performance at moderate system chip count and price.
3. The multiple chip design using general purpose control and data flow chips which results in high performance and a flexible design with somewhat higher chip count than the custom multi-chip approach of case 2 but similar system costs due to the use of high volume, general purpose chips.

The conclusion drawn in the above presentation is that multiple chip bipolar microprocessor designs should be either a custom chip set optimized for a particular instruction set or they should be designed using standard data flow chips and bipolar ROMs and TTL for the control section. This is due to the availability of a broad line of high performance bipolar ROMs and TTL logic which allows construction of powerful microprocessor control logic sections using a few standard chips and therefore greatly reduces the cost effectiveness of generalized microprogram control logic chips.

MACROLOGIC*—Versatile functional blocks for high performance digital systems

by KRISHNA RALLAPALLI and PETER VERHOFSTADT
Fairchild Camera and Instrument Corporation
Mountain View, California

INTRODUCTION

Evolution of bipolar technology in recent years has enabled semiconductor manufacturers to increase performance and packing density to the extent that meaningful standard functions of LSI complexity can be defined and built. MACROLOGIC is a powerful approach to accomplish this.

Technologically one of the densest junction isolation processes available and generous use of so-called non-standard circuit approaches have been applied. High performance is obtained because of shallow structures, extensive use of Schottky clamping, minimizing the nodal voltage swings, charge injection of switch transistors, etc.

Table I summarizes the features of the technology used. Aoplying this technology as a tool to improve the performance as well as the cost-effectiveness of digital systems, a set of standard functional blocks has been defined for applications in computers, control and communications systems.

FUNCTIONAL DESCRIPTIONS

This paper describes in detail a set of five parts using the MACROLOGIC approach. The user can select the required functions to implement his particular architecture. All parts have the following common features making them architecturally compatible:

(a) They are optimized for microprogrammed control.
(b) A 4-bit slice implementation is chosen and functions can be expanded to handle larger word lengths with few or no extra components.
(c) The devices are provided with three state outputs wherever appropriate, so that bus organized systems are easily realized.

TABLE I—MACROLOGIC Technology Features

- Delays of 4 *ns* per gate
- Speed-power product <8 *pJ*
- Complexities of 200-250 gates/chip
- Packing density of 50 gates/mm^2
- TTL compatible inputs/outputs
- Standard up-to 24-Pin DIP packages
- High drive capability for bus organized systems

(d) All operations occur synchronously with a clock.
(e) Any device can be addressed as a source of data by activating Output Enable (OE) and as a destination of data by activating the Execute (Ex) input.

Arithmetic logic register stack

The Arithmetic Logic Register Stack (ALRS) is designed to implement general registers in high performance programmable digital systems. As shown in Figure 1, it consists of a 4-bit ALU, 8 word by 4 bit RAM with latched outputs, instruction decode network, control logic and an output register. The ALU implements 8 arithmetic and logic functions where one 4-bit operand is supplied from an external source (input data bus) and the second 4-bit operand is supplied internally from one of the 8 RAM words. The selected operation is performed on the operands and the result is loaded into the same RAM location and simultaneously loaded into the output register which is made available through three-state buffers (output data bus). An active LOW Output Enable (OE) input controls these buffers. The instruction bus for the ALRS consists of two fields, A and I; A_0 through A_2 indicate the selected register and I_0 through I_2 specify the desired function to be performed. Thus, the ALRS provides 8 registers (R_0 through R_7) and 8 different

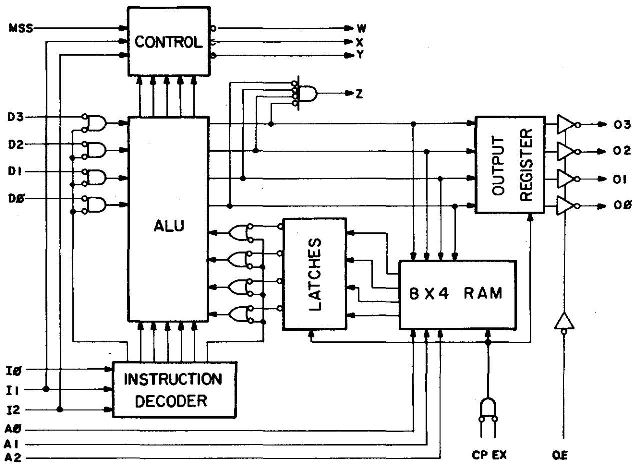


Figure 1—ALRS block diagram

* Trademark of Fairchild Camera and Instrument Corporation.

TABLE II—ALRS *I*-Field Assignment

I_2	I_1	I_0	INSTRUCTION	COMMENTS
L	L	L	(Rx) plus Bus plus 1→Rx	Add with carry
L	L	H	(Rx) plus Bus→Rx	Add
L	H	L	(R)x* Bus→Rx	Logical AND
L	H	H	Bus→Rx	Load Rx
H	L	L	(Rx)→Bus	Read Rx
H	L	H	(Rx) + Bus→Rx	Logical OR
H	H	L	(Rx) ⊕ Bus→Rx	Exclusive OR
H	H	H	$\overline{\text{Bus}}$→Rx	Load Complement

operations may be performed on any of these registers. The I_0-I_2 inputs are decoded by the instruction decode network to generate the necessary control signals for the ALU. (Table II lists *I*-Field code assignments.) The ALU also generates and transmits to the control logic Carry Out, Carry Propagate and Carry Generate, Negative status and Overflow status. The status Zero (Z) is generated and directly outputted. The control logic operates on these Status Signals (except Z) as a function of I_0-I_2 and a control MSS and generates three device outputs W, X and Y. The W output always represents the Carry Output from a slice. However, X and Y outputs represent Negative and Overflow for the most significant slice and represent Carry Propagate and Carry Generate for the remaining slices of an array. A high level on the MSS input declares the most significant slice in an array of ALRS's. All except the most significant device, should have a low level (ground) on MSS input.

Execution of an instruction is controlled by the clock when EX is low. The Instruction Bus and Data Bus are enabled when the Clock is high. Results are written into the RAM when Clock is low and are loaded into the output register on the low to high Clock transition.

The I_0 serves a dual purpose; during arithmetic instructions it is used as the carry input and for non-arithmetic instructions it serves as an instruction input. This is possible because only the two arithmetic instructions require a carry.

P-Stack

The P-stack is a 16 word by 4 bit "Push Down-Pop Up" Program Stack. It is designed to implement Program Counter (PC) and return address storage facilities in high performance programmable digital systems or can also be used as a 16-level general purpose stack.

It consists of an input multiplexer, a 16×4 RAM with latched outputs, an incrementor, control logic, Stack Pointer (SP), stack limit monitors and output buffers (See Figure 2).

When the device is initialized, the main PC will be the top location of the Stack. As new program counter values are "pushed" onto the Stack (Call Operation) all previous counter values move down one level. The top location of the Stack will be the current PC. After 15 entries, the original PC will be at the bottom or last location of the Stack giving a 15 level nesting capability. Information may also be "popped" from the Stack (Return Operation) bringing the most recent PC to the top of the Stack.

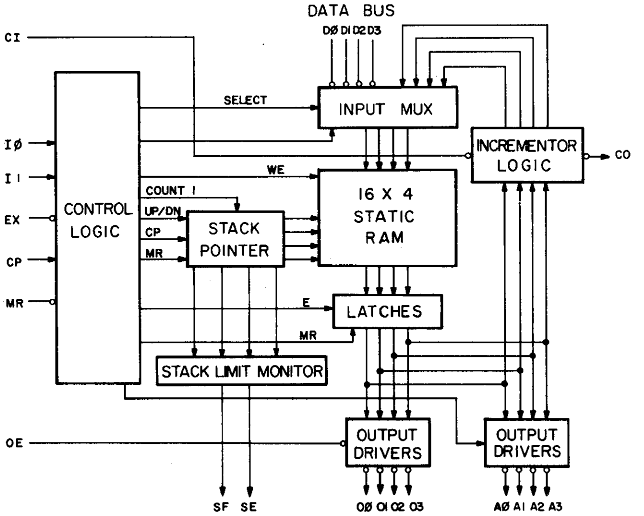


Figure 2—P-Stack block diagram

The P-stack executes 4 instructions: Return, Branch, Call and Fetch as specified by a 2-bit instruction (see Table III). A 4-bit input bus allows data to be loaded from an external source into the current PC. A 4-bit Address bus (A-bus) provides the current PC value as an output; in addition, this data is also available on a second 4-bit bus (O-bus) to allow effective address calculations relative to the program counter. Iterative instruction fetch can be accomplished by optional increment control of the current program counter via the CI input. The data inputs of the RAM are derived from the Data Bus (D-bus) or the incrementor, as selected by the input multiplexer.

The address for the RAM is obtained from the Stack Pointer (SP) which generates an incremented, decremented or unchanged address as a function of the instruction. The output of the RAM is stored in output latches. The O-bus is derived from the output latches and enabled by the active LOW Output Enable (OE) input. The A-bus is also derived from the output latches; it is enabled internally during the Fetch Instruction. Execution of instructions is controlled by the Execute (EX) and Clock (CP) inputs.

Operation of the active level LOW Master Reset (MR) causes the SP to be reset to the main PC (top of the Stack) and that RAM location is cleared to all zeros. The Stack Empty (SE) output will go LOW. This operation will

TABLE III—P-stack Instructions

I_1	I_0	INSTRUCTION	COMMENTS
L	L	SP-1→SP	Return (Pop)
L	H	D-bus→PC	Branch
H	L	SP + 1→SP, D-bus→PC	Call (Push)
H	H	PC→A-bus, PC + 1→PC	Fetch

override other inputs. In the event that the Stack becomes fully loaded, the Stack Full (SF) output goes LOW. If an additional Call Operation is performed after SP has reached $(1111)_2$, SP will increment to $(0000)_2$, the contents of that location will be written over, the Stack Empty (SE) will go LOW and SF will go HIGH (wrap around operation). When the top-most location is selected corresponding to SP= $(0000)_2$, as for example, after Master Reset (MR), the Stack Empty (SE) output is LOW. An additional Return Operation under these conditions forces SP= $(1111)_2$, causing SE to go HIGH and SF to go LOW (again wrap around operation).

Data path switch

The Data Path Switch (DPS) is an advanced two port multiplexor with two 4-bit input ports (D-bus and K-bus) and a 4-bit output port (O-bus). It has a 5-bit instruction bus (I-bus). The device consists of a data routing network, a control block section and output buffers (See Figure 3). The DPS can perform 32 different instructions as determined by the 5 I-lines (see Table 4 for a listing and description). The DPS is a completely combinatorial network without registers; it therefore does not have a clock input. The device always looks at the data bus, so the inputs are always open. Whether the result is being used is determined by the Output Enable (OE). The DPS not only can selectively gate one of two 4-bit ports onto the 4-bit output port, but also perform functions such as shifting and sign extending. Its typical use is to close the data path loops around arithmetic logic networks such as Arithmetic Logic Register Stack described before.

It also has many features normally associated with True Complement, One/Zero generators. It can generate true complement outputs of the input ports or the outputs can be forced to all ones or all zeros. In addition to all ones and all zeros, it can provide 0001, 1000 and 1110 as constants at the output. In arrayed operation, these constants can be used for incrementation, byte sign masking and decrementation.

Shift linkages left out (LO), right out (RO), left in (LI) and right in (RI) are available as individual inputs and outputs for complete flexibility in handling expansion and end around shifts.

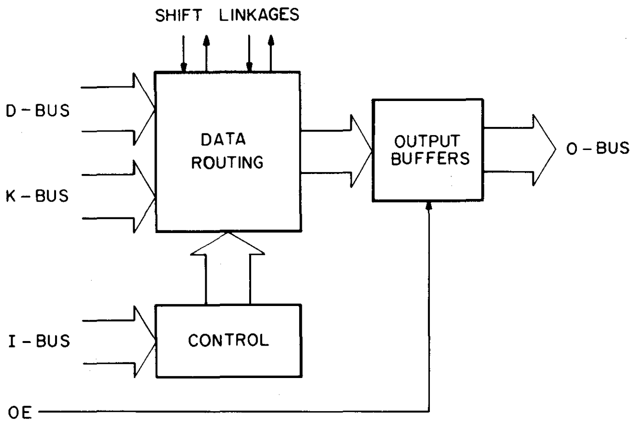


Figure 3—DPS block diagram

TABLE IV—DPS Instructions

I_4	I_3	I_2	I_1	I_0	INSTRUCTION	COMMENTS
L	L	L	L	L	1111 → OUTPUT	BYTE MASK
L	L	L	L	H	0000 → OUTPUT	
L	L	L	H	L	1110 → OUTPUT	−2 CONSTANT
L	L	L	H	H	1111 → OUTPUT	
L	L	H	L	L	D → OUTPUT	'OR' BYTE
L	L	H	L	H	0000 → OUTPUT	MASK D-BUS
L	L	H	H	L	D → OUTPUT	'AND' BYTE
L	L	H	H	H	1111 → OUTPUT	MASK D-BUS
L	H	L	L	L	1000 → OUTPUT	BYTE SIGN
L	H	L	L	H	0000 → OUTPUT	MASK
L	H	L	H	L	K → OUTPUT	'AND' BYTE
L	H	L	H	H	1111 → OUTPUT	MASK K-BUS
L	H	H	L	L	D → OUTPUT	LOAD BYTE
L	H	H	L	H	K → OUTPUT	
L	H	H	H	L	0001 → OUTPUT	+1 CONSTANT
L	H	H	H	H	0000 → OUTPUT	
H	L	L	L	L	RI → OUTPUT & LO	K-BUS SIGN EXTEND
H	L	L	L	H	K3 → LO & K → OUTPUT	
H	L	L	H	L	RI → OUTPUT & LO	D-BUS SIGN EXTEND
H	L	L	H	H	D3 → LO & D → OUTPUT	
H	L	H	L	L	RI → O_0, DO → O_1, D_1 → O_2, D2 → O_3, D3 → LO	SHIFT LEFT D-BUS
H	L	H	L	H	R1 → O_0, KO → O_1, K_1 → O_2, K2 → O_3 & K3 → LO	SHIFT LEFT K-BUS
H	L	H	H	L	LI → O_3, D3 → O_2, D2 → O_1, D1 → O_0 & DO → RO	SHIFT RIGHT D-BUS
H	L	H	H	H	D3 → O_3, D3 → O_2, D2 → O_1, D1 → O_0, DO → RO	SHIFT RIGHT ARITH D-BUS

(Continued on next page)

TABLE IV—DPS Instructions [CONTINUED]

I_4	I_3	I_2	I_1	I_0	INSTRUCTION	COMMENTS
H	H	L	L	L	LI → O_3, K_3 → O_2, K2 → O_1, K_1 → O_0, & KO → RO	SHIFT RIGHT K-BUS
H	H	L	L	H	K_3 → O_3, K_3 → O_2, K2 → O_1, K_1 → O_0 & KO→RO	SHIFT RIGHT ARITH K-BUS
H	H	L	H	L	K → OUTPUT	'OR' BYTE MASK
H	H	L	H	H	0000 → OUTPUT	K-BUS
H	H	H	L	L	$\overline{D}$ → OUTPUT	LOAD BYTE
H	H	H	L	H	$\overline{K}$ → OUTPUT	COMPLEMENT
H	H	H	H	L	NOT USED	
H	H	H	H	H	NOT USED	

Data Access Register

The Data Access Register (DAR) is designed to implement 16 instructions (see Table IV) which are suitable for memory address arithmetic and manipulation. It consists of a 4-bit adder, three 4-bit registers (R_0, R_1 and R_2), an output register and associated control logic (see Figure 4). It has two output ports the A-bus and the O-bus. Both ports are driven by three state buffers with individual output enables (OE1 and OE2). Carry In (CI) and Carry Out (CO) signals are available for expansion. One 4-bit operand for the adder is

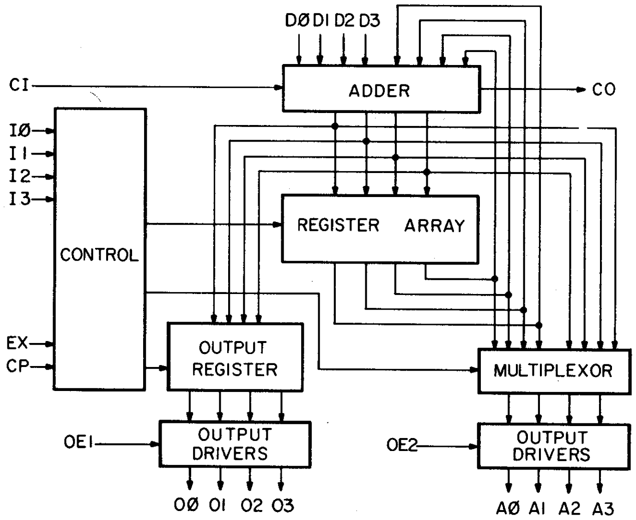


Figure 4—DAR block diagram

TABLE V—DAR—Instructions

I_3	I_2	I_1	I_0	INSTRUCTIONS	COMMENTS
L	L	L	L	R_0 → A-BUS, R_0 + D-BUS → R_0	R_0 is Updated By Adding With BUS
L	L	L	H	R_0 + D-BUS → A-BUS R_0 + D-BUS → R_0	Same as Above Except Updated Value is the Address
L	L	H	L	R_0 → A-BUS, R_0 + D-BUS → R_1	Update R_1 With the Sum of R_0 and BUS
L	L	H	H	R_0 + D-BUS → R_1 R_0 + D-BUS → A-BUS	Same as Above Except Updated Value is the Address
L	H	L	L	R_0 → A-BUS R_0 + D-BUS → R_2	Update R_2 with the Sum of R_0 and BUS
L	H	L	H	R_0 + D-BUS → A-BUS R_0 + D-BUS → R_2	Same as Above Except the Updated Value is the Address
L	H	H	L	R_1 → A-BUS R_1 + D-BUS → R_1	R_1 Supplies the Address and then Update R_1
L	H	H	H	R_1 + D-BUS → R_1 R_1 + D-BUS → A-BUS	Update R_1 and Updated Value is the Address
H	L	L	L	R_2 → A-BUS D-BUS → R_2	Use R_2 for the Address and Update R_2 from the BUS
H	L	L	H	D-BUS → A-BUS D-BUS → R_2	Update R_2 from the BUS and Updated Value is the Address
H	L	H	L	R_0 → A-BUS D-BUS → R_0	Use R_0 for Address and then Update R_0 from the BUS
H	L	H	H	D-BUS → A-BUS D-BUS → R_0	Update R_0 from the BUS and Updated Value is the Address
H	H	L	L	R_2 → A-BUS R_2 + D-BUS → R_2	R_2 Supplies the Address and then Update R_2
H	H	L	H	R_2 + D-BUS → R_2 R_2 + D-BUS → A-BUS	Update R_2 and the Updated Value is the Address
H	H	H	L	R_1 → A-BUS D-BUS → R_1	Use R_1 for Address and Update R_1 from the BUS
H	H	H	H	D-BUS → A-BUS D-BUS → R_1	Update R_1 from the BUS and Updated Value is the Address

always supplied by the D-bus while the second operand is obtained from one of the three registers. Independent of the destination register (R_0, R_1 or R_2), the result of an operation is always loaded into the output register from which the O-bus is derived. The A-bus is derived from two selectable sources; one of the three registers can supply its unmodified contents to the A-bus while the same register is being updated, or the updated result can be gated on the A-bus. In a typical application, the register utilization in the DAR may be as follows: R_0 is the program counter (PC), R_1 is the stack pointer (SP) (for memory resident stacks) and R_2 contains the operand address. For an instruction fetch, PC can be gated on the A-bus while it is being incremented (i.e., D-bus$=1$). If the instruction fetched calls for an effective address for execution, which is displaced from the PC, the displacement can be added to the PC and loaded into R_2 during the next microcycle.

R-Stack

The R-stack is a high speed 64-bit Read/Write Random Access Memory organized as 16 words by 4 bits. When the R-stack receives a LOW on the Execute (Ex) and Clock (CP) inputs, the instruction bits I_1, I_2, I_3 and I_4 select one of sixteen 4-bit words (see Figure 5). If the instruction bit I_0 is at a HIGH level, the contents of the selected word is non-destructively read out and presented to the output register. On the LOW to HIGH Clock transistion the output register is loaded with this data.

For a write operation, I_0 and Ex must be LOW. If this is the case, then while CP is LOW, data is written into the selected location. If the input data change during the period when CP, Ex and I_0 are LOW, the contents of the selected address will follow the changes (non-ones-catching) provided the set up time criteria are met. On the LOW to HIGH transition of the CP, the information again is loaded into the output register. A three state Output Enable (OE) controls the output buffers.

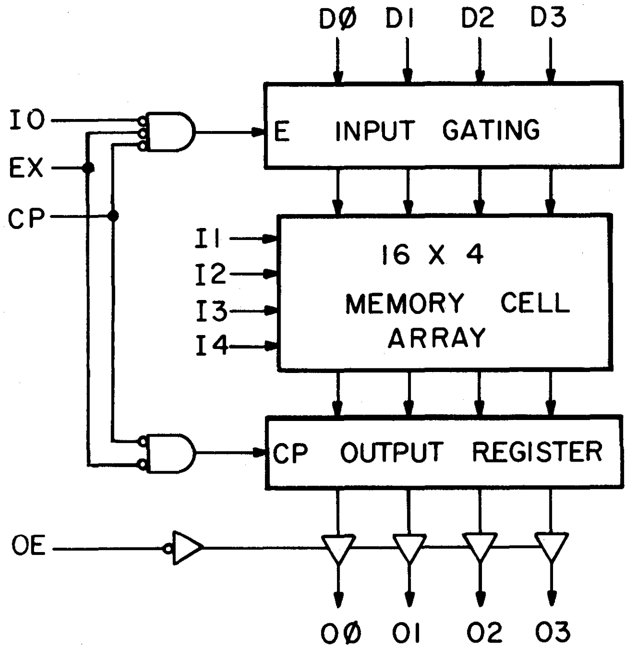


Figure 5—R-Stack block diagram

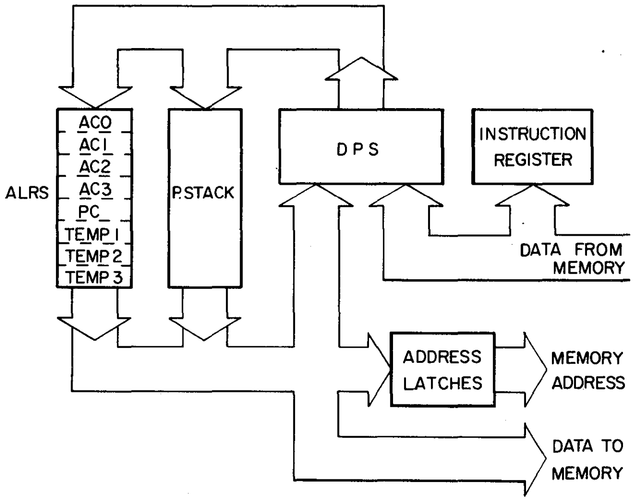


Figure 6—Data path for a hypothetical 16-bit processor

TYPICAL APPLICATIONS

One of the many possible applications for the class of components presented in this paper is to implement data paths for emulating existing instruction sets using microprogrammed control. Emulators using such high-speed, complex functional blocks are attractive because they offer improved cost/performance while retaining software compatibility with the target machine. For example, consider a 4-accumulator fixed word length 16 bit processor that uses two's complement arithmetic. It has a 16-word hardware stack for sub-routine return address as well as for general purpose stack use. Figure 6 shows a possible data path for this architecture. It has a memory reference instruction format as shown in Figure 7.

The two-bit index field specifies 4 addressing modes: base page, PC relative, AC2 relative and AC3 relative. For relative mode addressing, the 8 bit displacement field is treated as a signed number in two's complement notation. Figure 8 shows the micro sequence to implement a macro instruction fetch. Experience indicates that macro instructions can be fetched, interpreted and executed in a relatively small number of microcycles allowing sub-microsecond realizations.

One of the simplest microword formats for this kind of

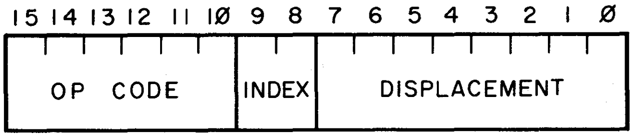


Figure 7—Memory reference instruction format

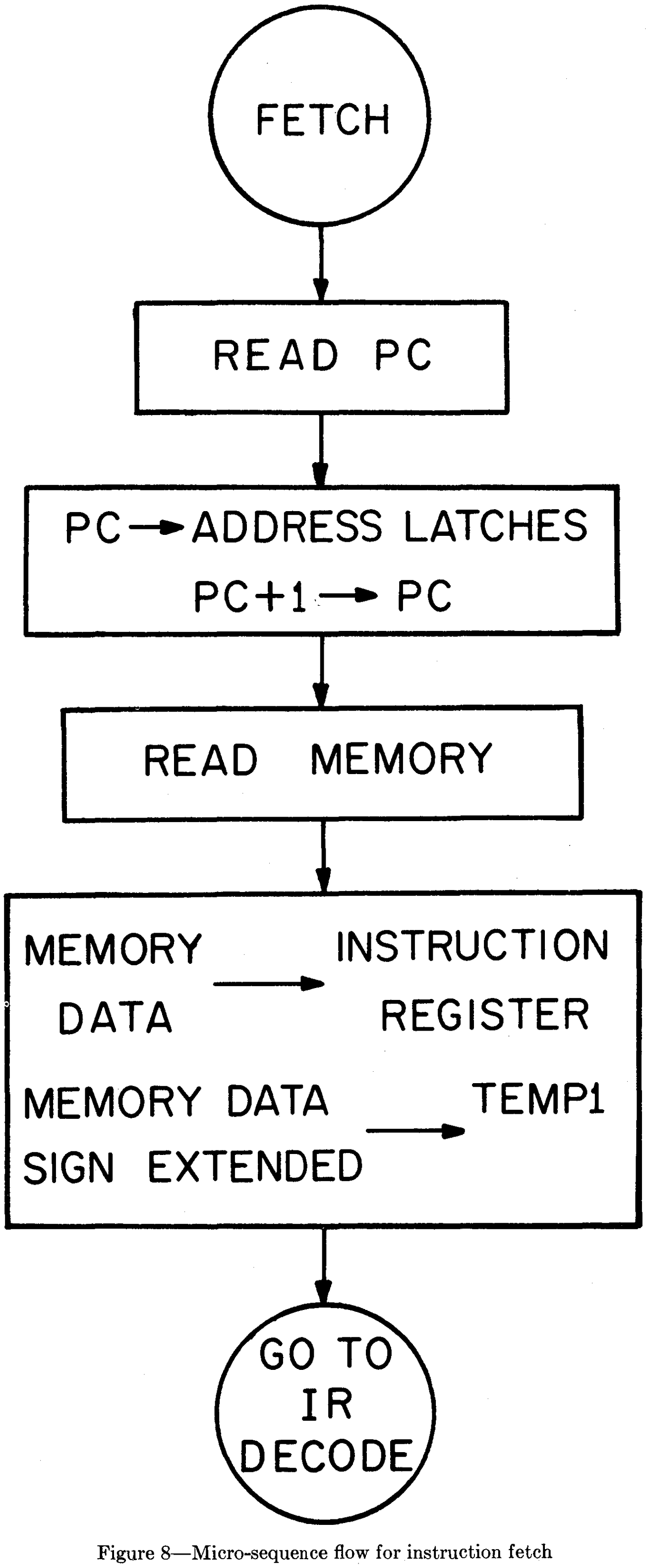

FETCH
READ PC
PC → ADDRESS LATCHES
PC+1 → PC
READ MEMORY
MEMORY DATA → INSTRUCTION REGISTER
MEMORY DATA SIGN EXTENDED → TEMP1
GO TO IR DECODE

Figure 8—Micro-sequence flow for instruction fetch

organization is based on the concept of Register Transfer Modules. There is one source of data and one or more destinations. During one clock cycle data will move from source to destination. The data can be operated on (according to the instruction tables) during such a transfer. Figure 9 is a possible microword format. The source field can be decoded to activate Output Enable (OE) inputs of the functions and the destination field can be decoded to activate the Execute (Ex) inputs. The function field drives the *I* lines of the devices described.

SUMMARY AND OUTLOOK

The functional elements described in this paper are essentially LSI building blocks for high-performance data path imple-

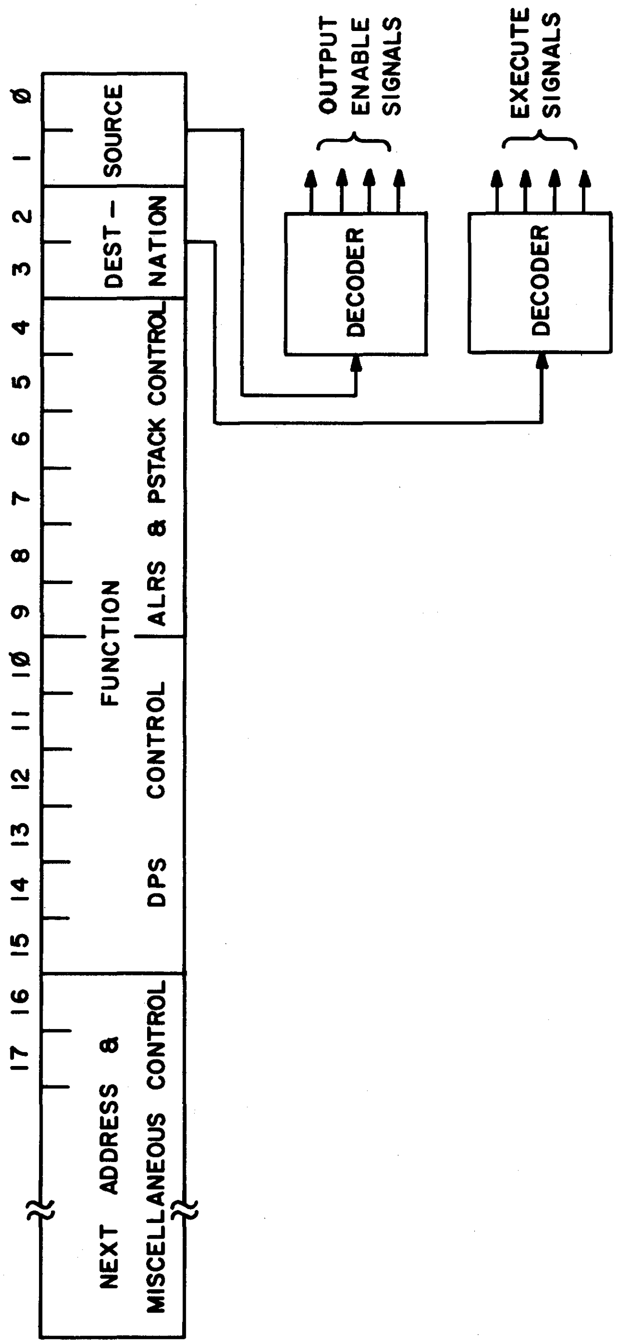


Figure 9—Microword format

mentation, allowing the construction of microprogrammable processors and controllers with clock rates of up to 10 MHZ.

Of equal importance as the datapath elements are of course the tools available to implement the control sections of the processor. For this purpose a series of compatible bipolar RAM's, ROM's and PROM's have been developed and the development of a micro-program control unit is well on its way to completion.

Further elements in the MACROLOGIC family for peripheral control applications are either available (e.g., FIFO, CRC) or in development.

ACKNOWLEDGMENTS

The authors would like to express their appreciation for the continuous support for their work from Dr. T. A. Longo, Vice President and Group General Manager of Fairchild Camera and Instrument Corporation as well as for the superb performance of the development groups involved in the effort.

Architecture of microcontroller system

by MICHAEL LICCARDO
Scientific Micro Systems
Mountain View, California

SYSTEM OVERVIEW

A microcomputer designed for control

The SMS MicroController is a microcomputer designed for control. It features:

Execution speed

- 300 nanosecond instruction execution time.
- Direct address capability—up to 4096 16-bit words of program memory.
- Eight 8-bit general purpose registers.
- Simultaneous data transfer and data edit in a single instruction cycle time.
- n way branch or n entry table lookup in two instruction cycle times.
- MicroController instructions operate with equal speed on 1-bit, 2-bit, 3-bit, 4-bit, 5-bit, 6-bit, 7-bit, or 8-bit data formats.

The MicroController instruction set features control oriented instructions which directly access variable length input/output and internal data fields. These instructions provide very high performance for moving and interpreting data. This makes the MicroController ideal in switching, controlling, and editing applications.

Interface simplicity and expandability

- Direct connection to TTL (3-state) I/O (Open Collector outputs are optional).
- I/O expandable to 224 connection points with storage buffer at each point.
- User defined data flow direction with each group of 8 I/O points.

External device signals may be accessed with minimal interface circuitry. The MicroController input/output system provides a direct register interface to external devices. Unlike classical minicomputer bus structures, external devices do not require the logic for providing addresses to the input/output system. The address of an external device is determined programmatically within the MicroController.

Direct processing of external data

Data from external devices may be processed (tested, shifted, added to, etc.) without first moving them to internal storage. This is because its I/O system appears to the MicroController as a set of internal registers. In fact, the entire concept is to treat data at the I/O interface no differently than internal data. This concept extends to the software which allows variables at the input/output system to be named and treated in the same way as data in storage.

Separate program storage and data storage

The storage concept of the MicroController is to separate program storage from data storage. Program storage is implemented in read-only memory in recognition of the fact that programs for control applications are fixed and dedicated. The benefits of using read-only memory are that great speeds may be obtained at lower cost than if read/write memory were used, and that program instructions reside in a non-volatile medium and cannot be altered by system power failures. Data storage for the MicroController is implemented with read/write memory because data in control and other real time applications is dynamic and variable.

High density packaging and reliable operation

- The MicroController is implemented completely with LSI circuits.
- The MicroController CPU consists of a single integrated circuit.
- Single +5.0 volt power supply operation.

The MicroController is provided packaged on one of four basic boards. The smallest packaging scheme is the System 10 which is 6.875 inches by 2.675 inches. This board can accommodate CPU, 2K words of program storage, and 32 I/O points. The largest package, the System 40, is 6.875 inches by 13.475 inches and accommodates a fully expanded system consisting of CPU, 4K words of program storage, 224 I/O points and 256 bytes of read/write data storage.

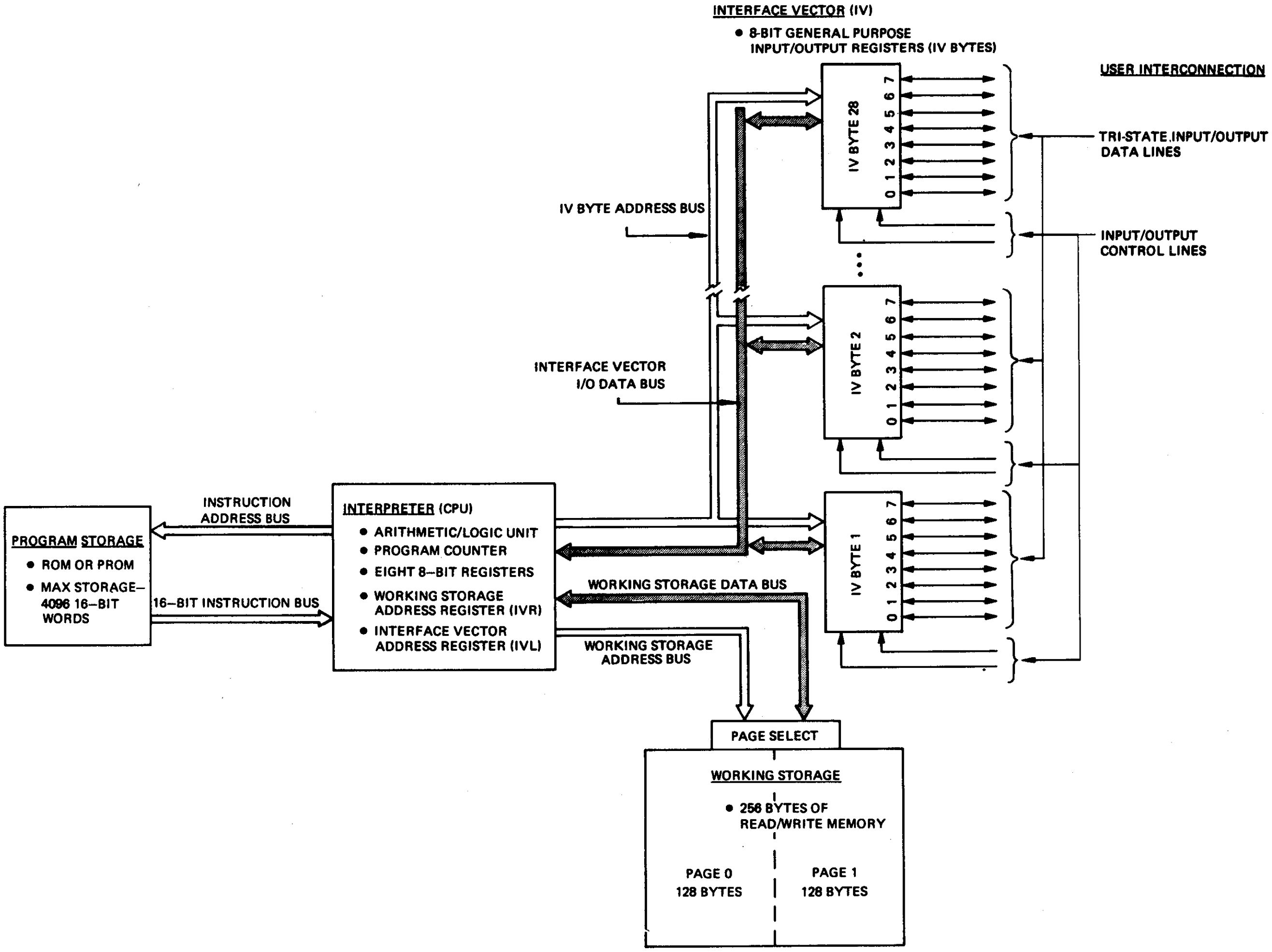


Figure 1—Microcontroller system diagram

MicroController functional components

The MicroController is a complete microcomputer system consisting of:

- A central processing unit called the Interpreter.
- Read-only program storage.
- Optional read/write data storage called Working Storage with variable field address of from 1 to 8 bits.
- A complete input/output system called the Interface Vector.

The MicroController System is shown in Figure 1.

Figure 2 illustrates the MicroController architecture. The MicroController CPU contains an Arithmetic Logic Unit (ALU), Program Counter, Interface Vector Address Register (IVL), and Working Storage Address Register (IVR). Eight 8-bit general purpose registers are provided, including seven working registers and an Auxiliary register which performs as a working register and also provides an implied operand for many instructions. The MicroController registers are shown in Figure 2 and are summarized below:

Control Registers include:

Instruction—A 16-bit register containing the current instruction.
Program Storage Address (AR)—A 13-bit register containing the address of the current instruction being accessed from Program Storage.
Program Counter (PC)—A 13-bit register containing the address of the next instruction to be read from Program Storage.
IV Byte Address (IVL)—An 8-bit register containing the address of the current byte being accessed from the Interface Vector. IVL is under program control.
Working Storage Address (IVR)—An 8-bit register containing the address of the current byte being accessed from Working Storage. IVR is under program control.

Data Registers Include:

Working Registers (WR)—Seven 8-bit registers for data storage.
Overflow (OVF)—A 1-bit register that retains the most significant bit position carry from ALU. Arithmetically treated as 2°.
Auxiliary (AUX)—An 8-bit register. Source of implied operand for arithmetic and logical instructions. May be used as a working register.

A crystal external to the CPU is used to generate the CPU system clock. The CPU provides eight instructions.

The 16-bit MicroController instructions are stored in 512 to 4096 words of read-only Program Storage. Program Storage can be implemented with either mask coded ROMs (Read-Only Memory) or PROMs (Programmable Read-Only Memory).

The input/output system, called the Interface Vector, serves as the data path over which information is transferred into and out of the MicroController. The basic elements of the Interface Vector are:

- The general purpose 8-bit input/output registers, or Interface Vector (IV) Bytes, whose tri-state data path serves as the connection points to the user system.
- The IVL register which contains the address of the IV Byte currently being accessed.
- Variable field selection which permits 1 to 8-bit field access of a selected IV Byte in a single instruction.

The Interface Vector eliminates the need for costly interface logic and presents a simple, well-defined interconnection point to the user system.

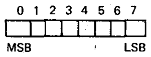
Working Storage is available as an option that provides 256 bytes of read/write memory for program data or input/output data buffering. Working Storage consists of:

- 256 8-bit bytes of read/write memory organized as two pages (banks), Page 0 and Page 1, of 128 bytes each.
- The Working Storage address register, IVR which holds the address of the byte currently accessed in either Page 0 or Page 1, depending on the state of the Page Select Register.
- The Page Select Register, addressed through IVR, is a single bit register used to select Page 0 or Page 1 of Working Storage.
- Variable Field Select which permits 1 to 8-bit field transfers to or from an addressed Working Storage byte in a single instruction.

MICROCONTROLLER INSTRUCTION SET

The MicroController has a repertoire of eight instructions which allow the user to test input status lines, set or reset output control lines, and perform high-speed input/output data transfers. All instructions are 16 bits in length. Each instruction is executed completely in 300 nanoseconds.

Data is represented as an 8-bit byte; bit positions are numbered from left to right, with the least significant bit in position 7.



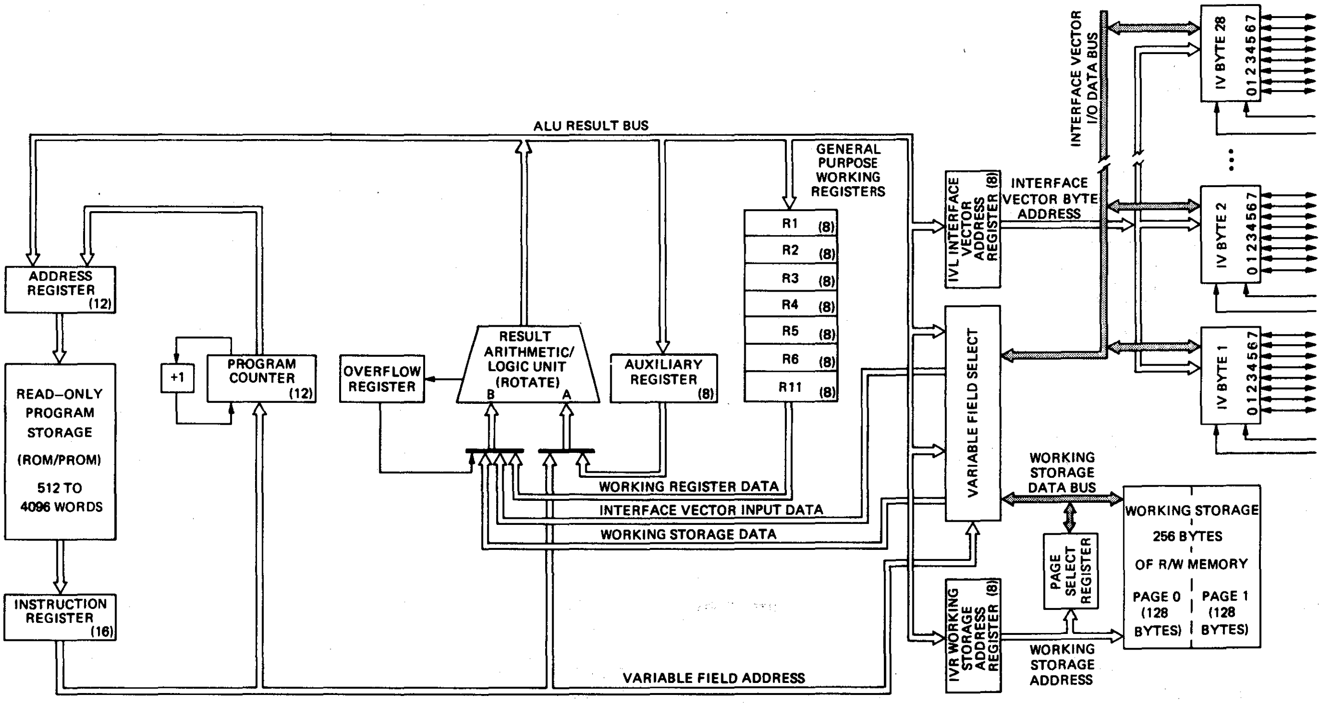


Figure 2—Microcontroller architecture

Within the Interpreter, all operations are performed on 8-bit bytes. The Interpreter performs 8-bit, unsigned, 2's complement arithmetic.

Instruction formats

The general MicroController instruction format is:

Instruction Formats

0 1 2	3 4 5 6 7 8 9 10 11 12 13 14 15
Op Code	Operand(s)

Table I contains a summary of the MicroController instruction set and description of the operand fields.

All instructions are specified by a 3-bit Operation (Op) Code field. The operand may consist of the following fields: Source (S) Field, Destination (D) Field, Rotate (R) Field, Length (L) Field, Immediate (I) Operand Field, and (Program Storage) Address (A) Field.

The instructions are divided into five format types based on the Op Code and the form of the operand(s).

TYPE I

0 1 2	3 4 5 6 7	8 9 10	11 12 13 14 15
Op Code	S	R	D

OPERATIONS

MOVE AND
ADD XOR

TYPE II

0 1 2	3 4 5 6 7	8 9 10	11 12 13 14 15
Op Code	S	L	D

OPERATIONS

MOVE ADD
AND XOR

TYPE III

0 1 2	3 4 5 6 7	8 9 10 11 12 13 14 15
Op Code	S	I

OPERATIONS

XEC XMIT
NZT

TYPE IV

0 1 2	3 4 5 6 7	8 9 10	11 12 13 14 15
Op Code	S	L	I

OPERATIONS

XEC XMIT
NZT

TYPE V

0 1 2	3 4 5 6 7 8 9 10 11 12 13 14 15
Op Code	A

OPERATIONS

JMP

TABLE I—MicroController Instruction Summary

OPERATION	FORMAT	RESULT	NOTES
MOVE	C S L D	Content of data field addressed by S, L replaces data in field specified by D, L.	If S and D both are register addresses then L specifies a right rotate of L places applied to the register specified by S.
ADD		Sum of AUX and data specified by S, L replaces data in field specified by D, L.	
AND		Logical AND of AUX and data specified by S, L replaces data in field specified by D, L.	
XOR		Logical exclusive OR of AUX and data specified by S, L replaces data in field specified by D, L.	
XMIT		The literal value I replaces the data in the field specified by S, L.	If S is IV or WS address then I limited to range 00-37. Otherwise I limited to range 000-377.
NZT	C S I	If the data in the field specified by S, L equals zero, perform the next instruction in sequence. If the data specified by S, L is not equal to zero, execute the instruction at address determined by using the literal I as an offset to the Program Counter.	If S specifies an IV or WS address then I is limited to the range 00 - 37. I is limited to the range 000 - 377 otherwise.
XEC	C S L I	Perform the instruction at address determined by applying the sum of the literal I and the data specified by S, L as an offset to the Program Counter. If that instruction does not transfer control, the program sequence will continue from the XEC instruction location.	The offset operation is performed by reducing the value of PC to the nearest multiple of 32 (if I : 00 - 37) or 256 (if I : 000 - 377) and adding the offset.
JMP	C I	The literal value I replaces contents of the Program Counter.	I limited to the range 00000 - 07777.

Instruction fields

Op code field—3-bit field

The Op Code field is used to specify one of eight MicroController instructions.

OP CODE OCTAL VALUE	INSTRUCTION		RESULT
0	MOVE	S,L,D	(S)→D
1	ADD	S,L,D	(S) plus (AUX)→D
2	AND	S,L,D,	(S)∧ (AUX)→D
3	XOR	S,L,D	(S) ⊕ (AUX)→D
4	XEC	I,L,S or I,S	Execute instruction at current PC offset by I+(S)
5	NZT	I,L,S or I,S	Jump to current PC offset by I if (S)≠0
6	XMIT	I,L,S or I,S	Transmit literal I→S
7	JMP	A	Jump to program location A

S,D fields—5-bit fields

The S and D fields specify the source and destination of the operation defined by the Op Code Field. The Auxiliary Register is the implied source for the instructions ADD, AND and XOR which require two source fields. That is, instructions of the form:

ADD X,Y

imply a third operand, say Z, located in the Auxiliary Register so that the operation which takes place is actually X+Z, with the result stored in Y. This powerful capability means that three variables are referenced in 300 nanoseconds.

0_8-17_8 is used to specify one of seven working registers (R1-R6, R11), Auxiliary Register, Overflow Register, IVL and IVR registers.

OCTAL VALUE	
00	Auxiliary Register
01	R1
02	R2
03	R3
04	R4
05	R5
06	R6
07	IVL Register—IV Byte address register—Used as a D field only, or S field in XMIT instruction.
10	OVF-Overflow register—Used as an S (source) field only.
11	R11
12	Unassigned
13	Unassigned
14	Unassigned
15	Unassigned
16	Unassigned
17	IVR Register—Working Storage address register—Used as a D field only, or S field in XMIT instruction.

20_8-27_8 is used to specify the least significant bit of a variable length field within the IV Byte selected by the address in the IVL register. The length of the field is determined by L.

OCTAL VALUE	
20	Field within selected IV Byte; position of LSB=0
21	=1
22	=2
23	=3
24	=4
25	=5
26	=6
27	=7

30_8-37_8 is used to specify the least significant bit of a variable length field within the Working Storage Byte selected by the address in IVR Register. The length of the field is determined by L.

OCTAL VALUE	
30	Field within selected W.S. Byte; position of LSB=0
31	=1
32	=2
33	=3
34	=4
35	=5
36	=6
37	=7

L/R field—3-bit field

The L/R field performs one of two functions, specifying either a field length (L) or a rotation (R). The function it actually does specify for a given instruction depends upon the contents of the S and D fields:

A. When both S and D specify registers, the R field is used to specify a right rotation of the data specified by the S field. (Rotation occurs on the bus and not in the source register.)

B. When either or both the S and D fields specify and IV or Working Storage Byte, the L field is used to specify the length of the field (within the byte) accessed, as shown below:

OCTAL VALUE
0—Field length=8 bits
1—Field length=1 bit
2—Field length=2 bits
3—Field length=3 bits
4—Field length=4 bits
5—Field length=5 bits
6—Field length=6 bits
7—Field length=7 bits

I field—5/8-bit field

The I field is used to load a literal value (a binary value contained in the instruction) into a register, IV or Working Storage Byte or to specify the low order bits of the Program Counter.

The length of the I field is based on S field:

A. When S specifies a register, the literal I is an 8-bit field (Type III format).

B. When S specifies an IV or Working Storage Byte, the literal I is a 5-bit field (Type IV format).

A field—13-bit field

The A field is a 13-bit Program Storage address field. In current systems, however, only 12 bits are used, resulting in storage capacity of 4096 instructions.

Register operations

When a register is specified as the source, and an IV or Working Storage field as the destination, the least significant bits of the operation (MOVE, ADD, AND, XOR) result are stored. The operation is performed on the entire 8-bit source for a MOVE, or between the 8-bit AUX and the source register for ADD, AND, XOR operations. The least significant bits of the result are stored in the IV or Working Storage field specified in the instruction.

When an IV or Working Storage field of one to eight bits is specified as the source, and a register as the destination, the 8-bit result of the operation (MOVE, ADD, AND, XOR) is stored in the register. The operations ADD, AND, XOR actually use the IV or Working Storage data field (1-8 bits) with leading zeros to obtain 8-bit source data for use with the 8-bit AUX data during the operation.

Because IVL and IVR registers can be specified as destination fields only, (see description of S, D fields), operations involving IVl and IVR as sources are not possible. For example, it is not possible to increment IVR or IVL in a single instruction, and the contents of IVL or IVR cannot be transferred to a working register, IV Byte, or Working Storage location.

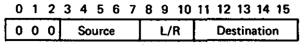
The OVF (Overflow) Register only can be used as a source field; therefore, it cannot be set or reset in a single instruction.

Instruction descriptions

The following instruction descriptions employ MCMAC (the MicroController Machine Compiler, described in a later section) programming notation. This notation varies somewhat from the instruction descriptions provided earlier. Thus, for example, explicit L field definition as shown is not required by MCMAC for machine instructions; MCMAC creates appropriate variable field addresses from the information contained in the Data Declaration statements provided by the programmer at the beginning of his program.

The MicroController instruction set is described below with examples illustrating instruction use.

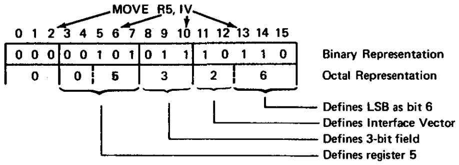
MOVE S,D or
MOVE S (R), D

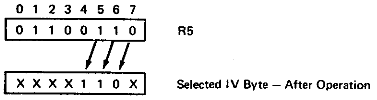


OPERATION: (S) → (D)

DESCRIPTION: Move data. The contents of S are transferred to D; the c· ntents of S are unaffected. If both S and D are registers, R specifies a right rotate of the source data during the move. Otherwise, L is implicit and specifies the length of the source and destination fields. If the MOVE is between an IV Byte and a Working Storage Byte, an 8-bit field is always moved.

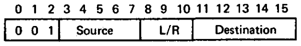
EXAMPLE: Store the least significant 3 bits of register 5 (R5) in bits 4, 5 and 6 of the IV Byte addressed by IVL register.





Note: X's in the IV Byte denote bits unaffected by the MOVE operation.

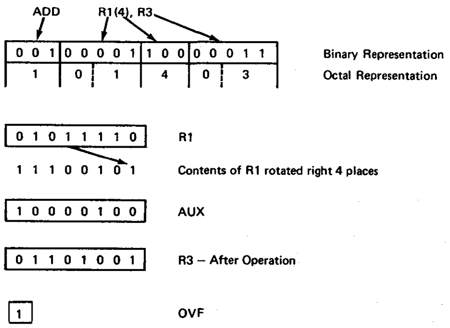
ADD S, D or
ADD S (R), D



OPERATION: (S) plus (AUX) → D; set OVF if carry from most significant bit occurs.

DESCRIPTION: Unsigned two's complement addition. The contents of S are added to the contents of the Auxiliary Register (which is the implied source). The result is stored in D; OVF is set. If both S and D are registers, R specifies a right rotate of the source (S) field before the operation. Otherwise L is implicit and specifies the length of the source and destination fields. S and AUX are unaffected unless specified as the destination.

EXAMPLE: Add the contents of R1 (rotated 4 places) to AUX and store the result in R3.



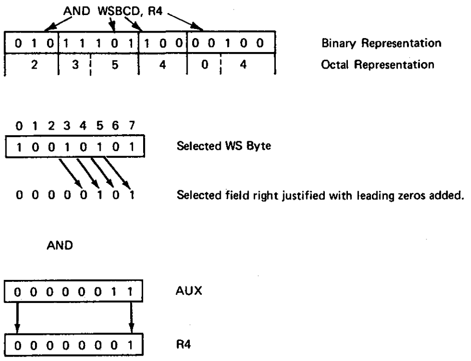
AND S, D or
AND S (R), D

0 1 2	3 4 5 6 7	8 9 10	11 12 13 14 15
0 1 0	Source	L/R	Destination

OPERATION: (S) ∧ (AUX) → D

DESCRIPTION: Logical AND. The AND of the source field and the Auxiliary Register is stored into the destination. If both S and D are registers, R specifies a right rotate of the source (S) field before the AND operation. Otherwise L is implicit and specifies the length of the source and destination fields. S and AUX are unaffected unless specified as a destination.

EXAMPLE: Store the AND of the selected Working Storage Byte and AUX in R4. The Working Storage Byte field is called WSBCD and is 4 bits long and located in bits 2, 3, 4 and 5.



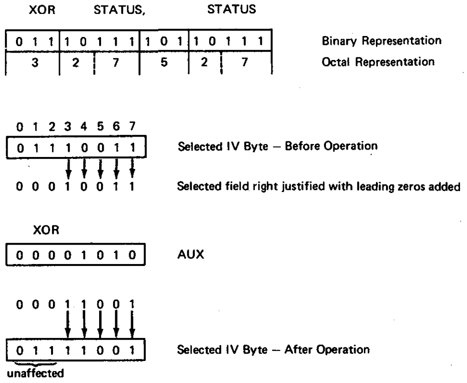
XOR S, D or
XOR S (R), D

0 1 2	3 4 5 6 7	8 9 10	11 12 13 14 15
0 1 1	Source	L/R	Destination

OPERATION: (S) ⊕ (AUX) → D

DESCRIPTION: Exclusive OR. The exclusive OR of the source field and the Auxiliary Register is stored in the destination. If both S and D are registers; R specifies a right rotate of the source (S) field before the XOR operation. Otherwise L is implicit and specifies the length of the source and destination fields. S and AUX are unaffected unless specified as a destination.

EXAMPLE: Replace the selected IV Byte field with the XOR of that field and AUX. The IV Byte field is called STATUS and is 5 bits in length and located in bits 3, 4, 5, 6 and 7.



XEC I(S)

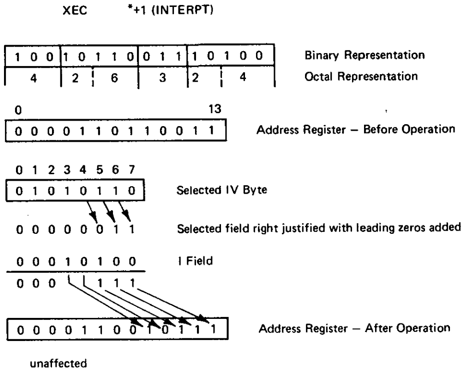
0 1 2	3 4 5 6 7	8 9 10 11 12 13 14 15
1 0 0	Source	I Field

0 1 2	3 4 5 6 7	8 9 10	11 12 13 14 15
1 0 0	Source	Length	I Field

OPERATION: Execute instruction at (Address Register) offset by (S) + I.

DESCRIPTION: Execute the instruction at the address determined by replacing the low order bits of the Address Register (AR) (which contains the current value of the Program Counter) with the low order bits of the sum of the literal I and the contents of the source field. If S is a register, the low order 8 bits of AR are replaced; if S is an IV or Working Storage Byte, the low order 5 bits of AR are replaced, resulting in an execute range of 256 and 32 respectively. The Program Counter is not affected unless the instruction executed is a JMP or NZT (whose branch is taken).

EXAMPLE: Execute a JMP in a table of JMP instructions determined by the value of the selected IV Byte field. The table follows immediately after the XEC instruction and the IV field is called INTERPT and is a 3 bit field located in bits 4, 5 and 6.



ADDRESS	INSTRUCTION
0000110110011	XEC *+1 (INTERPT)
0000110110100	JMP A1
• • •	
0000110110111	JMP A3
• • •	
0000110111011	JMP A7

JMP Table

JMP A3 is executed because IV field INTERPT = 3

XMIT I, S

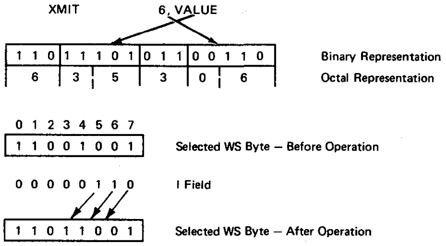
0 1 2	3 4 5 6 7	8 9 10 11 12 13 14 15
1 1 0	Source	I Field

0 1 2	3 4 5 6 7	8 9 10	11 12 13 14 15
1 1 0	Source	Length	I Field

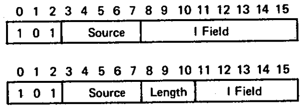
OPERATION: I → S

DESCRIPTION: Transmit literal. The literal field I is stored in S. If S is a register, an 8 bit field is transferred; if S is an IV or Working Storage Byte, up to a 5 bit field is transferred.

EXAMPLE: Store the bit pattern 110 in the selected Working Storage Byte field. The field name is VALUE and located in bits 3, 4 and 5.



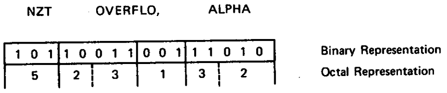
NZT S, I

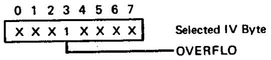


OPERATION: Non Zero Transfer If (S) ≠ 0, PC offset by I → PC; otherwise PC + 1 → PC.

DESCRIPTION: If the data specified by the S field is non-zero, replace the low order bits of the Program Counter with I. Otherwise, processing continues with the next instruction in sequence. If S is a register, the low order 8 bits are replaced; if S is an IV or Working Storage Byte, the low order 5 bits are replaced, resulting in an NZT range of 256 and 32 respectively.

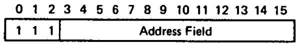
EXAMPLE: Jump to Program address ALPHA if the selected IV Byte field is non-zero. The field name is OVERFLO and it is a 1 bit field located in bit 3.





```
           ADDRESS                      INSTRUCTION

              •
              •
              •
           0000110110011                NZT OVERFLO, ALPHA
              •
              •
              •
ALPHA      0000110111010                Instruction
                  offset
```

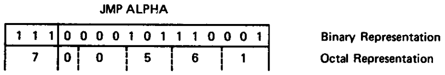
JMP A



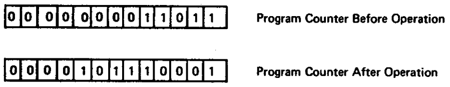
OPERATION: A → PC

DESCRIPTION: The literal value A is placed in the Program Counter and processing continues at location A. A has a range of 0 - 7777_8 in current systems (0 - 4095).

EXAMPLE: Jump to location ALPHA (0000101110001)



```
           ADDRESS                      INSTRUCTION

              •
              •
              •
           0000000011011                JMP ALPHA
              •
              •
              •
ALPHA      0000101110001                Instruction
```


0 0 0 0 0 0 0 0 1 1 0 1 1
Program Counter Before Operation
0 0 0 0 1 0 1 1 1 0 0 0 1
Program Counter After Operation

INPUT/OUTPUT SYSTEM

As seen from previous sections, the Interface Vector is the MicroController's input/output system. It provides a simple interconnection to the user status, control and data lines.

Addressing data on the interface vector

The Interface Vector is comprised of general purpose I/O registers called Interface Vector (IV) Bytes. In the present MicroController offering, the Interface Vector may consist of up to 28 IV Bytes.

As seen from Figure 2 the IVL register serves as the address register to the IV Bytes. In order for an instruction access (read or write) an IV Byte, the address of that byte must first be placed into the IVL register.

Thus, two instructions are required to operate on an Interface Vector byte:

XMIT ADDRESS, IVL
MACHINE INSTRUCTION

Once the IV Byte is selected (addressed) it will remain selected until the IVL register is loaded with another address. From the user's standpoint, however, all IV Byte outputs can be read by an external device regardless of whether they are selected or not.

Although the address range of IVL is $0\text{-}377_8$, only 28 IV Bytes are available on current system offerings. The addressing for the 28 IV Bytes is 01_8 to 34_8.

Electrical characteristics of the interface vector

Each IV Byte consists of 8 storage latches which hold data transferred between the Interpreter and the User System, 8 tri-state input/output lines and two input/output control lines, called Byte Input Control (BIC) and Byte Output Control (BOC) (Figure 3). The control lines functions are summarized in Table II.

READ/WRITE MEMORY

In MicroController applications, data may be stored in a read/write memory system called Working Storage.

TABLE II—Functions of the BIC and BOC Lines

CONTROL LINES		FUNCTION
BOC (low true)	BIC (low true)	
H	H	8 I/O lines in high impedance state—disable
L	H	8 I/O lines in output mode—8 bit storage latch data available in the output lines.
X	L	8 I/O lines in input mode—data can be read by Interpreter.

Table III contains a summary of the electrical characteristics of the IV Byte.

TABLE III—IV BYTE Terminal Electrical Characteristics

Characteristic	Symbol	Limits Min	Typ	Max	Units	Conditions
"1" Input Current*	$I_{1\,in}$			100	uA	$V_{1\,in}=5.5$ V
"0" Input Current*	$I_{\phi\,in}$			−800	uA	$V_{\phi\,in}=0.50$ V
"1" Input Voltage	$V_{1\,in}$	2		5.5	Volts	
"0" Input Voltage	$V_{\phi\,in}$	−1		0.8	Volts	
Input Clamp Voltage	$V_{c\,in}$			−1	Volts	$I_{\phi\,in}=-5$ma
High Output Voltage	V_{1out}	2.4			Volts	$I_{1out}=1$ma
Low Output Voltage	$V_{\phi out}$			0.5	Volts	$I_{\phi out}=-16$ma
Output Short Circuit Current	I_{so}	−20		−200	ma	$V_{\phi out}=$OV
Data Input Capacitance	C_{in}			12	pf	$V_{\phi\,in}=$OV

* Input current is always present regardless of the state of BIC and BOC.

Working Storage is accessed in much the same manner as IV Bytes. Figure 2 shows that IVR register is the Working Storage address register. It should also be noted from Figure 2 that a Page Select Register determines the page currently addressed by the IVR register. In order to access the Page Select Register, IVR must be set to 177_8, which is the address of the Page Select Register. Either a 1 or 0 can be transferred into bit 7 to select page 1 and page 0 respectively. Once the proper page is selected, IVR can be loaded with the address of the Working Storage Byte requiring access.

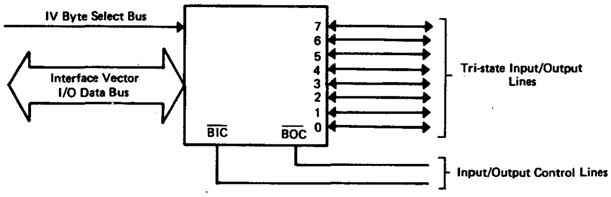

IV Byte Select Bus
Interface Vector I/O Data Bus
7
6
5
4
3
2
1
0
BIC
BOC
Tri-state Input/Output Lines
Input/Output Control Lines

Figure 3.1
IV BYTE PROVIDING DYNAMICALLY DEFINED DATA FLOW

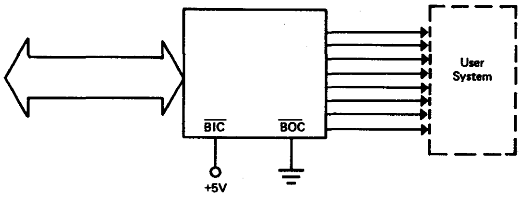


Figure 3.2
IV BYTE WIRED FOR USER OUTPUT ONLY

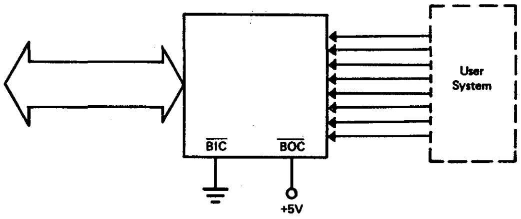


Figure 3—IV byte wired for input only

Because the two 128 byte pages of Working Storage are selected by the Page Select Register, the address loaded into IVR to access a byte in either page is identical 200_8-377_8. In effect, IVR holds the low order address bits and Page Select holds the high order address bit.

Operating on data in Working Storage requires two steps:

a. Selecting the 128 byte page which contains the data.
b. Accessing and operating on the actual data byte(s).

Page selection requires two instructions:

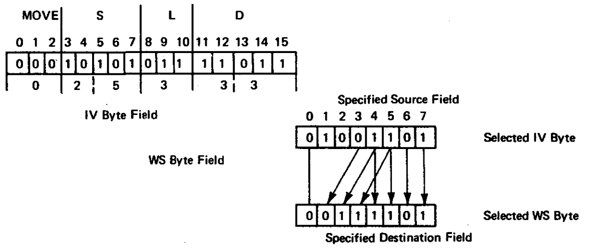
XMIT	177H, IVR	Enables Page Select Register
XMIT	PAGE, PSR	Selects Page

Thereafter all references to bytes within the selected page require two instructions:

XMIT ADDRESS, IVR Selects byte
MACHINE INSTRUCTION

When using instructions that involve the transfer of fields of less than 8 bits between an IV Byte and Working Storage, the following results should be noted.

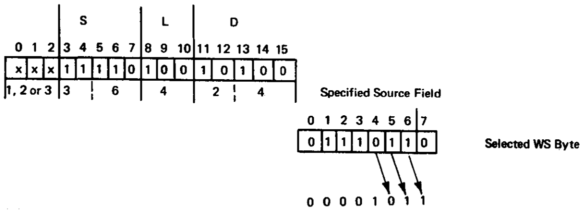
EXAMPLE: A MOVE instruction that specifies an IV Byte and Working Storage will have the following result:



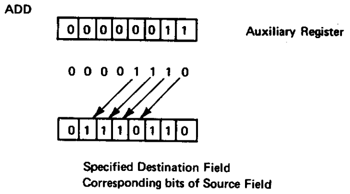
ACTION: The specified IV Byte source field is transferred into the specified Working Storage Byte field. The remainder of the destination byte is filled by the contents of the corresponding bit positions in the source bytes.

EXAMPLE: An ADD, XOR or AND instruction that specifies an IV Byte and Working Storage will have the following result:

ADD
AND
XOR



ADD



ACTION: The specified source field (right justified with leading zero's inserted) is added/anded/exclusive or'ed with the Auxiliary Register and the result placed in the destination field. The remainder of the destination byte is filled with the corresponding bit positions of the source byte.

EMMY—An emulation system for user microprogramming*

by M. J. FLYNN and C. NEUHAUSER
Stanford University
Stanford, California

and

ROBERT M. McCLURE
Palyn Associates, Inc.
San Jose, California

INTRODUCTION

A relatively unique emulation laboratory facility is being developed at Stanford University to support research in computer architecture and language processing. The center of this system is a "universal host" computer which is capable of emulating (or simulating) the behavior of many other computers.

The facility will serve three purposes. It will allow researchers to:

1. access a variety of computera rchitectures—facilitating inter-architecture comparisons—providing for the processing of archival code for obsolete computers;
2. analyze the effectiveness of various computer architectures and compilers through the use of "software probes";
3. develop new "soft" computer architectures which reflect the artifacts of specific higher level languages—ultimately each higher level language would have its own coded machine architecture dynamically loaded into a host system.

For some time microprogramming techniques have been used in the design of computer control units. [1,2,3] However, in the past the principal interest has centered around read-only-memory microprogram systems.

The introduction of very high speed, read-write storage based on large scale integration (both bipolar and fast MOS technologies) represented a significant change in the above environment. Now microprograms—and data—could be rapidly loaded into the "control storage" which we term "microstorage" here. The environment of the Seventies introduced the possibility that one "host" system could serve as an emulator for a wide variety of "image" systems. This led to the introduction of two new architectural concepts: the soft computer architecture; [4] and dynamic microprogramming. [5]

The soft computer architecture—as represented by the Nanodata QMI [6] machine and to the Burroughs B1700 [7] takes advantage of this fast read-write capability, coupling it with a number of innovative processor features including:

1. field handling and selection
2. high speed shifting ability
3. extensive bit testing
4. flexible specification of data paths (residual control)

Each of these features can be used effectively in implementing interpretive emulation processes.

A soft architecture is enhanced through a technique known as "dynamic microprogramming"—in which the read-write micromemory is identified as the primary storage media of the system as well as the medium which contains the emulator code. Such architectures are arranged to both execute microinstructions and fetch data out of this fast storage media. The "control storage" then becomes a microstorage which more closely resembles an explicit Cache than a simple ROM. The advantage of dynamic microprogramming is that data access times can be shortened by having the data present in this high speed storage media, thus resulting in improved system performance.

EMMY

Emmy is the name given to the processor which forms the nucleus of our facilities at Stanford. It was designed, with severe cost constraints, to be an efficient as well as unbiased host machine. The goal was for a CPU design that could fit on one large printed circuit board with an inherently high instruction processing rate. Further, the design would have to accommodate the flexibility required to emulate a variety of conventional machines as well as to allow the development of new, abstract, language oriented machines. As a result, EMMY is both a soft architecture and dynamically micro-

* The general architecture of EMMY was based on a series of studies conducted at the Johns Hopkins University and was supported, in part, by the U.S. Atomic Energy Commission under contract AT(11-1 3288).

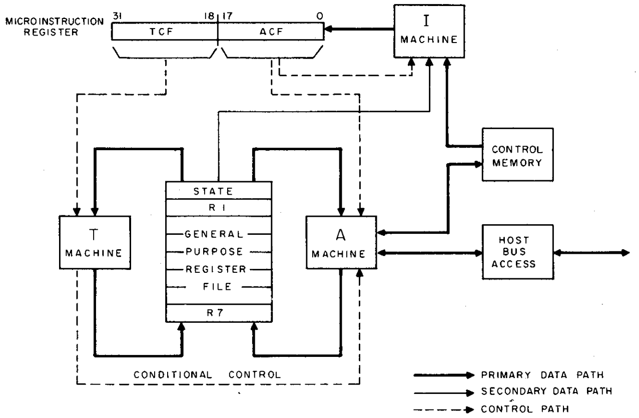


Figure 1—Structure of host machine

programmable. It is based upon a series of simulated systems which have been developed by our research group over the past several years. The EMMY machine is a 32 bit system which has 4,096 words (32 bit) of fast microstorage (access time 60 nanoseconds—cycle time less than 200 nanoseconds). The system is implemented using a very high speed technology—e.g., the switching technology has an internal cycle time of 25 nanoseconds. It is highly organized about LSI both in memory area and in processor implementation, and typically executes an instruction once every 200 to 400 nanoseconds. A strong influence on the design was the desire to minimize the amount of logic required to implement it, since cost and size were considered very critical.

An unusual feature, for a machine of this size, is that there exists a high degree of parallelism within the individual instructions. The host machine contains three separate, yet interdependent, finite state machines, each receiving control input from the current microinstruction and each controlling a resource associated with one class of instructions (Figure 1). These machines are designed as:

1. T-machine (controls functional resources),
2. A-machine (controls memory resources), and
3. I-machine (controls fetching of the next microinstruction).

Microinstructions in the host machine are formatted so that, in general, one half of the instruction (the T-control field or TCF) controls the T-machine and the other half (the A-control field or ACF) controls the A-machine. The I-machine may be controlled by either or both halves of the microinstruction.

Both the T- and A-machines manipulate data residing in the eight general purpose registers. The A-machine also moves data between micromemory and the registers and initiates communications with external memory units on the host bus. I-machine operation controls the fetching of the next microinstruction from micromemory. Host machine state information necessary to control the I-machine is contained in register 0 of the register file. Since this state register is directly accessable to the microprogrammer, flexible procedure oriented operations are possible.

Instruction set structure

Microinstructions (Figure 2) are 32 bits in length—the leftmost 14 bits, the TCF field, being dedicated to the control of the T-machine and the remaining 18 bits, the ACF field, being dedicated to the control of the A- and I-machines. Note that although there is a high degree of parallelism in these instructions, the TCF and ACF fields are vertically encoded independently of each other. The resulting microinstruction set is a relatively simple programming medium.

T-machine instructions

T-machine instructions are designed to provide the basic functional operations that the microprogrammer needs to emulate the functional and control aspects of a target machine. Instructions for the T-machine may be divided into the following classes:

1. logical,
2. arithmetic,
3. shift and rotate,
4. extended arithmetic, and
5. field insert and extract.

Instructions in the first four classes have a standard format which specifies opcode, subopcode, two register operands and indicates the possible use of immediate data. When immediate data is specified, the 18 bit field usually used to control the A-machine is expanded into a 32 bit quantity of immediate data. The extended arithmetic instructions subopcodes are designed to give the microprogrammer powerful single cycle operations with which to build complex target machine instructions, such as multiply and divide, by repetition.

Field insert and extract instructions are full word instructions which the microprogrammer may use to isolate and

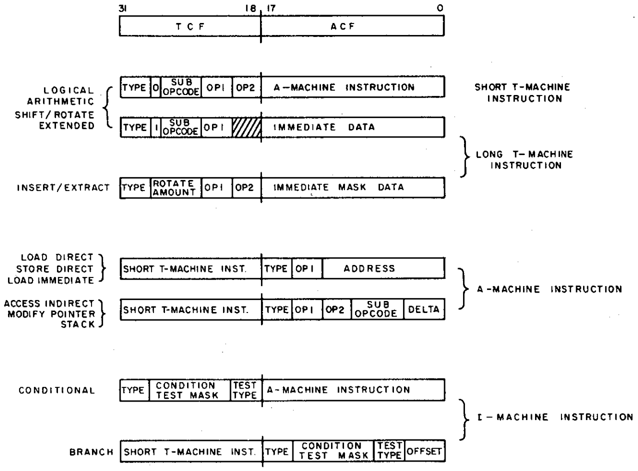


Figure 2—Structure of host machine instruction set

move fields of a data word residing in the registers. The insert instruction, for example, takes a word from one register, rotates it by a specified amount (0-31 bit positions) and places the result in a designated register under masking specified by the ACF field. This instruction is useful in breaking down target machine instructions and in matching host machine resources to target machine requirements when their word lengths differ.

A-machine instructions

A-machine instructions are used by the microprogrammer to access micromemory, manipulate address pointers, and communicate with external devices on the host bus system. A-machine instructions fall into the following classes:

1. move registers directly to and from micromemory,
2. load a register with immediate data,
3. access memory resources indirectly,
4. manipulate pointers, and
5. maintain stacks in micromemory

Access to external memory is designed so that once the operation is initiated the instruction address counter may continue to advance while awaiting the completion of the operation. This is an important source of parallelism in the emulation of instruction and data fetch in many target machines. A-machine stack operations allow the microprogrammer to access and maintain stacks in micromemory. Pointer manipulation instructions involve register incrementing, decrementing, addition, and conditional branching on results. Stack and pointer operations are particularly useful for operand indexing and sequencing of interpretive subroutines.

I-machine instructions

Fetching of the next microinstruction is controlled by the I-machine. Microinstructions are fetched sequentialy from micromemory unless the I-machine is specifically directed to fetch from a different location. Since the machine state, which includes the microinstruction address, is contained in one of the general purpose registers, the programmer may change the usual sequence by using the current microinstruction to modify the state register.

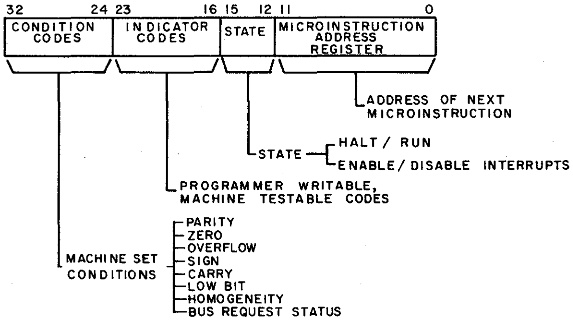


Figure 3—Layout of host machine state register

Within the state register is an eight bit condition code field representing various aspects of the previous T-machine operation (Figure 3). Instructions are provided to allow the microprogrammer to test these condition codes and control the operation of the A-and I-machines. These instructions are classified as:

1. conditional,
2. branching, and
3. looping

A conditional instruction is one in which the TCF field of the microinstruction specifies the testing of the condition codes and controls the subsequent execution of the A-machine. If the indicated condition is found to hold then the instruction for the A-machine, as specified in the ACF field, is executed, otherwise it is skipped. Using this facility the microprogrammer is able to specify conditional jumps, stacking operations, memory accesses and so forth.

A branch instruction may be specified in the A-machine control field (ACF) and allows the programmer to test the condition codes and perform a short relative jump from the current location based on the results. This instruction is used to provide control of the I-machine concurrently with T-machine operation.

Pointer modification instructions, which control the A-machine, may also provide looping capability. The results of each pointer modification operation may be tested for one of the common arithmetic conditions (e.g., less than zero), and the results of the test may cause a short relative jump. This instruction allows the microprogrammer to control repetitive operations such as normalize and multiply. In fact, the emulation of a target machine multiply instruction requires only one microinstruction since the extended arithmetic instruction "multiply step" and the looping instruction may be combined.

DATA FLOW DESCRIPTION

The general purpose register file (Figure 3) consists of seven 32 bit working registers and one status register. The status register (reg. 0) contains status and machine state information including the micromemory address pointer. The seven working registers are all full accumulators. There are two registers designated Register A and Register B at the input to the T-machine. These are temporary holding and shifting registers between the register file and the Arithmetic Logic Unit (ALU) in the T-machine. The ALU always accepts one operand from register A, and may accept the second operand from register B, the immediate field from the microinstruction, micromemory, or an outside resource. A multiplexer switch that gates the appropriate second operand to the ALU is also termed the "expansion unit" in that it can gate partial word operands left or right justified with zero or one fill in the remaining bits.

The ALU result is gated back to the appropriate (designated in the microinstruction) register in the register file. For arithmetic and logical operations a second cycle then gates the condition code into the correct field in register 0.

The Micro-Instruction Register holds the current microinstruction being executed. As noted earlier, each instruction is divided into two parts: T-control field (TCF) and the A-control field (ACF). The TCF controls data transformation resources and the ACF controls auxiliary operations such as loads, branches, and I/O operations. For some instructions the ACF may not be executed due to the result of the TCF execution satisfying a given condition. Immediate fields from Micro-Instruction Register may be gated to either the ALU or micromemory.

The micromemory address counter is located in the rightmost 12 bits of register 0.

All registers save those in the register file are transparent to the microprogrammer, although he should obtain a somewhat qualitative understanding of EMMY's architecture and its hardware operation.

REGISTER 0 (STATE REGISTER)

Register 0 contains 4 main fields. They are the Condition Code, Indicator Code, Machine State, and the Micromemory Address Counter.

The CCODE is set by arithmetic, logical and various internal operations. The CCODE comprises bits 31 through 24 of register 0. Bits 31 through 25 make the arithmetic condition code and are set only by arithmetic and logical operations. The significance of each bit is listed in Figure 3.

Bit 24 is set when micromemory is busy. The microprogrammer must test this bit to determine completion of memory cycle.

The indicator code, ICODE, (bits 23 through 16) is for programmer access only. Various TCF instructions may access this field for the purpose of setting flags or any other purpose the programmer deems reasonable (or unreasonable if he so desires). Thus the CCODE code is set by the machine while the ICODE is set by the programmer. However, both are testable by the branch and both are saved on an interrupt.

The Machine State is depicted by bits 15 through 12. The functions are indicated on Figure 3.

The Micromemory Address Counter, MAC, (bits 11 through 0) points to the next instruction to be fetched in the MIR from micromemory. After the MIR is loaded, the MAC is incremented, and the instruction execution begins.

ADDRESSING SCHEME

Certain devices within the EMMY and all external devices have an address assigned to them. All are connected to a common address and data bus. When an address is gated onto the address bus, each device looks to see if it is the device being selected. The following table lists all the internal device addresses. All other addresses are either unused or the address of an external device.

Address	Device
FF0000-FF0FFF	Micromemory
FF1000-FF1007	Register file
FE0000	Address display register
FE0001	Data display register
FE0002	Data/Address switch register
FE0003	Push button register
000000-03FFFF	Main memory addresses

INTERRUPTS

When the interrupt system is enabled and an interrupt is received, Register 0 is saved at a micromemory location (with the Micromemory Address Counter field incremented by (1) corresponding to the type of interrupt. Register 0 is then reloaded from an associated location and execution resumes. By reloading register zero, the programmer can (obviously) change all the information contained in register zero, that is, the condition and indicator codes, the machine state, and the Micromemory Address Counter.

A (partial) table of defined micromemory interrupt locations is listed below. Register zero is saved at the odd location (the listed address plus one) and is reloaded from the even location (the listed address).

Location	Interrupt Type
44	Console interrupt
46	Main memory interrupt
48	Console interrupt
4A	Block Transfer interrupt
4C	Bus time-out interrupt

PERFORMANCE

Microinstructions which reference other registers are executed in the 200-250 ns range with the exception of long shifts which require an extra 25 ns per bit shifted after 2 bits, extract-insert, which require long shifts, and a few of the extended instructions. Referencing micromemory requires an additional 200 ns. Referencing the external bus requires a varying amount of time depending on the unit referenced and the function performed. Memory activity specifically is conducted asynchronously, and the CPU need not wait for a memory read or write. Memory completion testing may be either explicit or implicit. A single statement of performance is difficult to make. A few examples might be most informative:

Example Timings of Emulated Instructions

Multiply—32×32=64 bit* (2's complement)	7.2 us
Divide—64/32=64 bit* (correct sign, 2's comp. rem.)	8.4 us
Binary to Decimal—20b to 7d* (unsigned)	5.0 us

Decimal to Binary—11d to 32b* (signed 2's comp., 360 style dec.)	9.2 us
MVC—360, 32 bytes non-aligned (assumes 1.0us main memory for both instructions and data)	12.8 us
AR—360 RR add (assumes 1.0us memory for instructions, regs. in micromemory)	7.4 us

IMPLEMENTATION

The natural choice of TTL was initially selected. It became apparent very early, however, that to meet the performance objective of an average of 200ns per microinstruction, it would be necessary to both use a large percentage of Schottky TTL and also provide two adder paths with attendant interlocking difficulties.

Since we have had extensive experience with high-speed current-mode logic, we took a second look at the cost and advantages of using MECL-10K logic. We found that the logic problems were simpler and the performance target could more easily be met. The additional care required in mechanical and electrical design did not prove to be a serious problem.

The resulting design requires a minimum number of logic design tricks. It is a straightforward synchronous design with a 25 ns clock. The T operations, A operations, and I operations are viewed as three cooperating finite state machines. Each of these three machines is implemented as a 16 state machine (although not all states are currently used in any of them). By implementing these three functions as independent, automata, simultaneous use of CPU resources is achieved with minimum difficulty. The bus control logic is independent of all of these and serves to further maximize overlap.

Physically, the CPU is on a single PC card of approximately 12″×15″. The micromemory and console logic each have a PC card.

CONCLUSIONS

EMMY is a low-cost, soft machine developed using high speed technology. This system has uniform 32 bit instructions and data paths.

The instruction format exhibits threefold parallelism: transformational specification, auxiliary (move and pointer handling) and an implied next instruction fetch. This parallelism together with fast native performance (200-400 nsec/instruction) produces respectable emulation capability across a variety of target machines.

REFERENCES

1. Wilkes, M. V., "The Best Way to Design an Automatic Calculating Machine," *Manchester University Computer Inaugural Conference*, 16-18, July 1951.
2. Husson, S. S., *Microprogramming Principles and Practice*, Prentice-Hall 1970.
3. Tucker, S. G., *Microprogram Control for System 360*, Vol. 6, No. 4, 1967.
4. Flynn, M. J. and M. D. MacLaren, "Microprogramming Revisited," *ACM National Conference Proceedings*, Vol. 22, Thompson Books, Washington, D.C. 1967, pp. 457-464.
5. Cook, R. and M. J. Flynn, "System Design of a Dynamic Microprocessor," *IEEE Transactions on Computers*, Vol. C-19, No. 3, pp. 213-222, March 1970.
6. Q M-1, Nanodata Corp., Buffalo, New York.
7. *B1700, Systems Reference Manual*, Burroughs Corp., Detroit, Michigan, 1972.

* not including main memory access and instruction interpretation

Instruction sequencing in microprogrammed computers

by LOUISE H. JONES*

University of Delaware
Newark, Delaware

INTRODUCTION

The purposes of this paper are to review the microinstruction sequencing capabilities of several microprogrammed computers; to determine whether these sequencing capabilities permit easy implementation of the control constructs of flowchartable program logic in modular microcode; and to present a set of microinstruction sequencing functions which will support "structured" microprogramming. Several microprogrammable mini- and microcomputers which provide the user with the means for implementing special purpose instruction sets have been introduced relatively recently.[1] However, the experiments by Weber[2] and Balzer[3] which demonstrated the possibilities for increasing computation speeds, decreasing main memory space usage, and easing the task of applications programming by means of special purpose instruction sets implemented in microcode were performed some time ago.

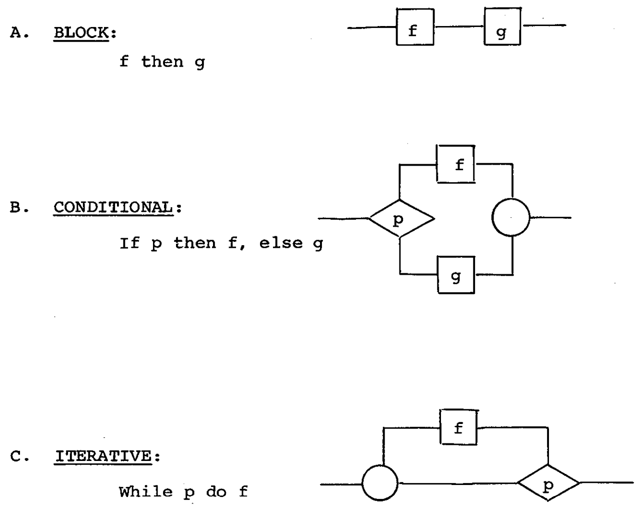
Any trend toward implementing more complex functions in microcode raises the question of whether the methodology of "structured programming" described by Dijkstra[4] and Mills[5] should be applied to microprogramming in order to manage the complexity of the programming task. This requires stepwise refinement of the function into subfunctions related by a limited number of control constructs until the subfunctions can be described easily in terms of modules of microcode. In order to test the microprogram, it is necessary that the behavior of the modules of microcode be defined independently of their context at the next higher level. In addition, a microprogrammed implementation of a special purpose instruction that is based on "context-free" modules of microcode will minimize the control memory requirements of the system by allowing each module of microcode to be called from several different locations in the microprogram.

The microinstruction sequencing capabilities of microprogram machines provide the basic mechanism for implementing various forms of program logic. Previous discussions[6] of microinstruction sequencing have been primarily concerned with differences in the number of branching conditions that can be implemented using various sequencing schemes. There has been little or no discussion of the relations between the sequencing capabilities of microprogrammable computers and the microcoded implementation of the control constructs either of structured programming or of other classes of program schema.[7]

TYPES OF INSTRUCTION SEQUENCING IN MICROPROGRAM PROCESSORS

Various microinstruction sequencing strategies have been implemented in contemporary microprogram machines. These strategies range from the explicit generation of the complete address of the next microinstruction using an address which is specified in the current microinstruction and possibly modified by the status of the machine (e.g., the IBM System/360[8]) to the implicit calculation of the next microinstruction address either by incrementing the contents of a microprogram address register or by incrementing the contents of an alternate microprogram address register (e.g., the Burroughs Interpreter[9]). In addition, the sequencing capabilities of microprogram machines depend on the design of the microinstruction. In vertically microprogrammed machines, each instruction usually controls a single operation and the address of the next microinstruction is obtained implicitly by incrementing the address of the current microinstruction unless the current microinstruction affects a conditional or unconditional BRANCH; in this case the address of the next microinstruction is generated explicitly from an address field in the instruction and the status of the machine (e.g., the Hewlett-Packard 21MX[10]). In contrast, in horizontally microprogrammed machines, each microinstruction controls multiple operations including the testing of appropriate conditions and microinstruction sequencing. Horizontal microinstructions usually specify, frequently implicitly, both the address of the next microinstruction to be executed on success and that of the instruction to be executed on failure of the test (e.g., the Naval Research Laboratory MCU[11]). Specific examples of microinstruction sequencing capabilities for several microprogrammable machines are given in the next section.

* Presently employed by E. I. du Pont de Nemours & Co., Wilmington, Delaware.



Note: f and g may be flow charts consisting of structures A., B., and C.

Figure 1—Basic control constructs for flowchartable program logic

IMPLEMENTATION OF THE CONTROL CONSTRUCTS OF STRUCTURED PROGRAMMING IN MICROCODE

The basic methodology of structured programming requires the stepwise refinement of flowchartable program logic using the basic sequential (BLOCK), conditional (IF-THEN-ELSE), and iterative (WHILE-DO) control constructs shown in Figure 1 where *f* and *g* may be single lines of microcode, straight line sequences of microcode, or any other structure defined recursively from the structures in Figure 1. Microcoded implementations of the control constructs of Figure 1 have been developed for the special cases where *f* and *g* are single microinstructions and straight line segments of microcode for several representative microprogram computers having both explicit and implicit implementations of microprogram sequencing functions. These implementations are schematic in the sense that the control function of the microinstruction is emphasized and the corresponding data transformation function is ignored. While the specific form of the schematic microcode is machine dependent, the following simple conventions have been used:

1. Register assignments are made from left to right.
2. ADR(*f*) denotes the control memory address of microinstruction *f*; this microinstruction may be denoted either by its function, *f*, or a label, ABC.
3. (AMPCR) denotes the contents of register AMPCR.
4. The meaning of the successor commands such as STEP, JUMP, CALL are machine dependent and are defined in Tables I, III, IV, VI, or in the case of the HP-21MX, in the description of the machine.
5. The schematic microinstructions for Machine V are written as IFETCH ops/EXECUTE ops where the IFETCH operations have the format (successor, ALTINSAR stack operations).

The Burroughs interpreter[9]

The Burroughs Interpreter has two different types of instructions, Type I and Type II, which differ both in function and in possible successors. Sequencing is defined implicitly using two pointers into control memory; these are called the microprogram count register (MPCR) and the alternate microprogram count register (AMPCR). Type I microinstructions contain pointers to horizontal control words stored in a "nanomemory"; there are eight possible successor commands for Type I instructions which are described in Table I. These include the standard STEP and SKIP commands, CALL and RETN commands which provide one level of subroutine capability, and EXEC which provides indirect addressing by allowing the single microinstruction at the address specified by (AMPCR)+1 to be executed out of sequence. Each nanoinstruction includes a test field and fields for specifying successors to the microinstruction for both success and failure of the test. In contrast, Type II microinstructions are used to load literals into various registers; the (implicit) successor command for these microinstructions is always STEP.

Although the sequencing commands of this machine permit easy implementation of the basic control constructs for flowchartable program logic when *f* and *g* are single lines of microcode; implementation of these constructs is much more difficult when *f* and *g* are straight line segments of microcode (macros) (see Table II). The first difficulty appears in the implementation of the IF-THEN-ELSE construct; different return mechanisms are required for the two macros *f* and *g*. This means either that there must be two copies of each macro in the control memory of that the set of all macros must be partitioned

TABLE I—Successor Commands for the Burroughs Interpreter[9]

Successor Command	Next Microinstruction (MPCR)	Next Alternate Microinstruction (AMPCR)
STEP	(MPCR)*+1	**
SKIP	(MPCR)+2	**
SAVE	(MPCR)+1	(MPCR)
CALL	(AMPCR)+1	(MPCR)
JUMP	(AMPCR)+1	**
RETN	(AMPCR)+2	**
WAIT	(MPCR)	**
EXEC	(AMPCR)+1***	**

* (MPCR) denotes "contents of microprogram count register."

** Denotes "no change."

*** EXEC causes a single microinstruction to be executed out of sequence; there is no change in (MPCR).

TABLE II—Microcoded Implementation of the Basic Control Constructs of Flowchartable Program Logic

Host Machine:	Burroughs Interpreter
Structure of f and g:	Straight line segments of microcode
Control Construct	Schematic Microcode*
f then g	ADR(f_1)−1=: AMPCR CALL ADR(g_1)−1=: AMPCR CALL Where f: f_1, STEP and where g : g_1, STEP ⋮ ⋮ f_n, JUMP g_m, JUMP
If p then f else g	ADR(f_1)−1=: AMPCR If p then CALL else STEP ADR(g_1)=: AMPCR CALL Where f: f_1, STEP and where g : g_1, STEP ⋮ ⋮ f_n, STEP g_m, JUMP (AMPCR)+2=: AMPCR JUMP
While ($\sim p$) do f	XXX: ADR(f_1)−1=: AMPCR** If ($\sim p$) then JUMP else STEP Where f: f_1, STEP ⋮ f_n, STEP ADR(XXX)−1=: AMPCR JUMP

* ADR (f_1) denotes the control memory address of microinstruction f_1, XXX is a label for a microinstruction.

** Much of the difficulty in implementing the WHILE-DO construct in "context-free" microcode results from the required STEP successor for microinstructions used to load literals.

into one subset that is called only on success of a test and a second subset that is called only on failure of a test. In either case there are difficulties. The second difficulty appears in the implementation of the WHILE-DO construct; here a specific return address must be embedded within the macro f. This means that the macros used in WHILE-DO constructs cannot be modular; there must be one copy of f for each WHILE-DO loop involving this function and the control memory will contain blocks of nearly identical microcode. It should also be noted that different return mechanisms must be embedded in f for use in the IF-THEN-ELSE and WHILE-DO constructs.

The Hewlett-Packard 21MX[10]

This machine is a vertically controlled microprogrammable minicomputer with encoded microinstructions. Normal microinstruction execution is sequential with branching capabilities derived from two types of instruction, CONDITIONAL JUMP and UNCONDITIONAL JUMP; the address of the microinstruction that is the target of the jump is specified explicitly in the instructions. In addition, returns from subroutines can be accomplished by means of a particular encoding ("RETN") of the "SPECIAL" field of the COMMON type of instruction; the implicit target address of the return is contained in the SAVE register which can be loaded only by the instruction which specifies an (unconditional) jump to a subroutine, JSB. The SAVE register is cleared by the "RETN" microoperation. Microinstruction sequencing is sequential unless a jump, subroutine jump, or subroutine return is specified.

Implementations of the control constructs for flowchartable program logic, using the sequencing capabilities of the HP21MX, are given in Table III. It is clear that the combination of the unconditional subroutine jump microinstruction (JSB) and the RETN microoperation support modular implementation of the BLOCK construct when f and g are straight line segments of microcode. However, the HP21MX does not have a conditional subroutine jump microinstruction and this means that a JMP instruction giving a specific return address must be added to f (as an unconditional jump) in order to implement the WHILE-DO construct. Similarly a JMP instruction specifying a return address must be added to the module of

TABLE III—Microcoded Implementation of the Basic Control Constructs of Flowchartable Program Logic

Host Machine: Hewlett-Packard 21MX
Structure of f and g: Straight line segments of microcode

Control Construct			Schematic Microcode*			
f then g		JSB**	ADR(f_1)			
		JSB	ADR(g_1)			
	Where f:	f_1		and where g:	g_1	
		⋮			⋮	
		f_n,	RETN		g_m,	RETN
If p then f else g		XXX:	JMP CNDX p ADR(f_1)***			
		XXX+1:	JSB ADR(g_1)			
	Where f:	f_1		and where g:	g_1	
		⋮			⋮	
		f_n				
		JMP	ADR(XXX+2)		g_m,	RETN
While p do f		XXX:	JMP CNDX p ADR(f_1)***			
	Where f:	f_1				
		⋮				
		f_n				
		JMP	ADR (XXX)			

* ADR(f_1) and ADR(XXX) denote the control memory addresses of microinstruction f_1 and the microinstruction whose label is XXX, respectively.
** JSB=Jump to subroutine, JMP CNDX=conditional jump, see Reference 10.
*** ADR(f_1) and ADR(XXX) must refer to the same 256 word control memory module.

microcode selected on success of the test in the IF-THEN-ELSE construct. Thus, the microinstruction sequencing capabilities of the HP21MX, like those of the Burroughs Interpreter, do not support implementation of the control constructs of structured programming in modular microcode.

The Argonne Microprocessor[12]

The Argonne Microprocessor (AMP) is an experimental horizontally controlled microprogrammable processor designed as a tool for research in microcontrol, including microsequencing, language processing, and processor design research. The microinstruction sequencing capabilities include incrementing the microinstruction address register (UMAR) by one or by two (unconditionally or if one of five bus conditions is satisfied), or jumping to a location specified by the jump address field in the current microinstruction (unconditionally or if one of five bus conditions is satisfied) or to the microinstruction specified by incrementing the instruction on the top of the microaddress stack. The default successor of the test of a condition is the microinstruction at (UMAR)+1. The sequencing capabilities of the Argonne Microprocessor are summarized in Table IV.

Implementations of the control constructs of structured programming using the sequencing capabilities of the AMP are given in Table V. The implementation of the BLOCK construct in modular microcode is straightforward as in the cases of the Burroughs Interpreter and the Hewlett-Packard 21MX. Implementation of the WHILE-DO construct requires that the address of the instruction before the appropriate jump instruction be stored because the return address mechanism of the AMP increments the address on the top of the microaddress stack. Microcoded implementation of IF-THEN-ELSE requires that a

TABLE IV—Sequencing Capabilities of the Argonne Microprocessor[12]

Successor Command	Next Microinstruction (UMAR)	Condition
STEP	(UMAR)*+1	
SKIP	(UMAR)+2	
SKPCD1	″	(B_i)**<0
SKPCD2	″	(B_i)=0
SKPCD3	″	(B_i)>0
SKPCD4	″	(B_i)=11 . . . 1
SKPCD5	″	(B_i)=ODD
JUMP	(JUMP ADDRESS FIELD)***	
JMPCD1	″	(B_i)<0
JMPCD2	″	(B_i)=0
JMPCD3	″	(B_i)>0
JMPCD4	″	(B_i)=11 . . . 1
JMPCD5	″	(B_i)=ODD
RETN	(MICRO ADDRESS STACK)+1****	

* (UMAR) denotes the contents of the micromemory address register.
** B_i denotes Bus 1 or Bus 2 depending on bit 72 of the control word.
*** JUMP ADDRESS FIELD refers to bits 1-11 of the control word.
**** The return address is pushed onto the μSTACK if bit 71 of the control word is set. RETN pops this stack.

TABLE V—Microcoded Implementation of the Basic Control Constructs of Flowchartable Program Logic

Host Machine: Argonne Microprocessor
Structure of f and g: Straight line segments of microcode

Control Construct	Schematic Microcode
f then g	JUMP (to ADR(f_1)),* PUSH JUMP (to ADR(g_1)), PUSH Where f: f_1 and where g: g_1 ⋮ ⋮ f_n, RETN g_m, RETN
If p then f else g	XXX: JMPCDX** (to ADR(f_1)), JUMP (to ADR(g_1)), PUSH Where f: f_1 and where g: g_1 ⋮ ⋮ f_n, JUMP (to ADR(XXX)+2) g_m, RETN
While p do f	———— STEP, PUSH JMPCDX** (to ADR(f_1)) Where f: f_1 ⋮ f_n, RETN

* The address of microinstruction f_1 is specified in the Jump Address Field of the current microinstruction.

** It is assumed that p corresponds to one of the conditions listed in Table IV. The testable conditions in the AMP are quite limited.

specific return address be embedded in the module of microcode entered on success of the test p but not in that entered on failure of p. The requirements for embedded return addresses in this construct are similar to those of the Burroughs Interpreter.

The Microprogrammed Univac C/SP Processor[13]

A horizontally controlled microprogrammed version of the Univac C/SP Processor has been described by Redfield.[13] The microinstruction sequencing capabilities of this processor were designed to support the interpretation of a machine instruction set and permit the next microinstruction address to be the contents of the main memory read data bus, the current microinstruction address plus one, the contents of the MARK register, or the address specified explicitly by a special eight bit field in the current microinstruction. Loading of the MARK register is controlled by the mark bit in the microinstruction; if this bit is set, the address of the current microinstruction plus one is loaded into the MARK register. Conditional branches are accomplished using the T-field. The microinstruction sequencing capabilities of the Univac C/SP processor are inadequate for implementation of the IF-THEN-ELSE structure shown in Figure 1 in context-free modules of microcode for the same reasons that the microinstruction sequencing capabilities of the Argonne Microprocessor are inadequate and will not be discussed further.

TABLE VI—Successor Commands for Machine V

Successor Command	Next Microinstruction (INSAR)	Next Alternate Microinstruction (ALTINSAR)	INSAR Stack Operations
True Successors			
STEP	(INSAR)***+1	*	**
SKIP	(INSAR)+2	*	**
SAVE	(INSAR)+1	(INSAR)+1	**
CALL	(ALTINSAR)	*	Push(INSAR)+1
JUMP	(ALTINSAR)	*	**
RETN	(TOS)****	*	Pop
WAIT	(INSAR)	*	**
False Successors			
STEP	(INSAR)+1	**	**
SKIP	(INSAR)+2	**	**

* For each true successor, the ALTINSAR stack pointer may be left unchanged (NOP), incremented (IAP), or decremented (DAP).

** Denotes "no change".

*** (INSAR) denotes the contents of the microprogram count register.

**** (TOS) denotes the contents of the top of the INSAR stack.

The INTEL 3001 Microprogram Control Unit[14]

Recently Intel has announced the INTEL 3001 Microprogram Control Unit which controls the microinstruction sequencing for the new, high-speed INTEL 3002 microprocessor. The INTEL 3001 permits explicit addressing of 512 microinstructions; the jump operation field in the current microinstruction specifies one of four unconditional or seven conditional address control functions which use selected bits of the current machine state (e.g., latch, flags, and accumulator bits) to compute the address of the next

TABLE VII—Microcoded Implementation of the Basic Control Constructs of Flowchartable Program Logic

Host Machine:	Machine V
Structure of *f* and *g*	Straight line segments of microcode
Control Construct	Schematic Microcode*
f then *g*	(STEP, NOP)/ADR(f_1)**=: ALTINSAR (CALL, NOP)/ADR(g_1) =: ALTINSAR (CALL, NOP)/_____
If *p* then *f* else *g*	(STEP, NOP)/ADR(f_1)=: ALTINSAR If *p* then (CALL, NOP) else (SKIP)/ADR(g_1)=: ALTINSAR (SKIP, NOP)/_____ (CALL, NOP)/_____
While *p* do *f*	(STEP, NOP)/ADR(f_1)=: ALTINSAR XXX: If *p* then (CALL/NOP) else (SKIP)/ADR(XXX)=: ALTINSAR (JUMP, NOP)/_____
Where *f*:	(STEP, NOP)/f_1 and where *g*: (STEP, NOP)/g_1 (RTN, NOP)/f_n (RTN, NOP)/g_m

* In Machine V the microinstruction IFETCH phase includes condition test, successor choice and ALTINSAR stack operations and is completed before the execution phase. Microcode is written IFETCH ops/EXECUTE ops with (Successor, ALTINSAR stack ops).

** ADR(f_1) denotes control memory address of the microinstruction with execution function f_1

Note: These modular implementations result largely from Machine V's capability for specifying all successors for instructions that load the ALTINSAR.

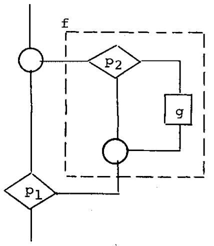
microinstruction from that of the current microinstruction. These functions include 16-way jump and test instruction. It is clear that the microinstruction sequencing strategy used in the INTEL 3001 does not permit implementation of the control constructs of flowchartable program logic in modular microcode. Furthermore, the implementation in microcode of a macroinstruction set or of a specific controller algorithm will be quite difficult, primarily because of the difficult translation from the control structures commonly used to express algorithmic tasks to those implemented as the microinstruction sequencing options of this machine.

MACHINE V, A MICROINSTRUCTION SEQUENCING SET FOR "STRUCTURED" MICROPROGRAMMING

None of the five microprogrammed machines described in the previous section have microinstruction sequencing functions which permit implementation of the structures of Figure 1 in "context-free" modular microcode. However, it is possible to design a set of microinstruction sequencing functions which permit implementation of the structures of Figure 1 in modular microcode not only for *f* and *g* being straight line segments of microcode but also for the general case that *f* and *g* are structures defined recursively from any of the structures in Figure 1. The machine with these sequencing capabilities is called Machine V.

The sequencing capabilities of Machine V were designed to permit implementation of flowchartable program logic using completely modular subroutines (no embedded return addresses or "tricky" sequencing instructions such as "AMPCR+2=:AMPCR") and a relatively low number of bits to control microinstruction sequencing. Machine V has two hardware stacks: the INSAR (*INS*truction *A*ddress *R*egister) stack, which is used to save return addresses for subroutine calls, and the ALTINSAR

A. Structure



B. Schematic Microcode (see Table VII for coding conventions)

```
    Address         Microinstruction
       1.   (STEP, NOP)/ADR(f1) =:  ALTINSAR
       2.  XXX:  If p, then (CALL, IAP) else (SKIP)/ADR(XXX)=: ALTINSAR
       3.   (JUMP, DAP)/ _____

      10.   f1:  (STEP, IAP)/ADR(g1) =:  ALTINSAR
 f    11.   yyy: If p2 then (CALL, IAP) else (SKIP)/ADR(yyy)=: ALTINSAR
      12.          (JMP, DAP)/ ___
      13.          (RTN, DAP)/ ___

 g    20.   g1:  (STEP, NOP)/ ___
      29.        (RTN, NOP)/ ___
```

Note: Both f and g are modular.

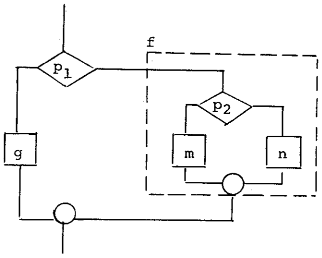
Figure 2—Implementation of nested while-do structures in modular microcode (Machine V)

(*AL*Ternate *INS*truction *A*ddress *R*egister) stack, which is used to eliminate repeated loading of identical alternate addresses. In Machine V fetching of microinstruction $i+1$ is done in parallel with execution of microinstruction i; the following sequence of events occurs during the fetch cycle: (1) fetch microinstruction $i+1$, (2) specify test condition, (3) choose successor microinstruction, and (4) perform ALTINSAR stack operations; it is assumed that the execution of microinstruction i is completed *before* the ALTINSAR stack operations associated with fetching microinstruction $i+1$ are performed.

The sequencing functions of Machine V are given in Table VI. If the test of a condition succeeds, one of seven successor microinstructions is selected and one of three ALTINSAR stack operations is performed. Only the STEP and SKIP successors can be specified for the unsuccessful test and no ALTINSAR stack operation can be executed. Seven bits of the microinstruction are required for these sequencing functions: 3 bits for the TRUE successors, 2 bits for the ALTINSAR stack operation, 1 bit for the FALSE successor, and 1 bit to negate the condition.

Implementations of the control constructs of Figure 1 using the sequencing functions of Machine V are given in Table VII for the special case where f and g are straight line segments of microcode. In addition, examples of the use of the ALTINSAR stack to implement nested WHILE-DO and IF-THEN-ELSE structures are given in Figures 2 and 3. Rules governing the use of the ALTINSAR stack pointer in modularized subroutines are: (1) the pointer must be at the same location when entering and leaving a subroutine, and (2) the ALTINSAR pointer must be incremented *before* loading the ALTINSAR stack. The sequencing capabilities of Machine V permit a subroutine to be called from several points in the program and, due to the INSAR and ALTINSAR stacks, permit the basic structures of flowchartable program logic embedded within each other (to the limits of the stacks) to be implemented in modular microcode. Machine V, unlike the machines discussed in the previous section, has a hardware control structure which supports the control structures which provide a basis for flowchartable program logic. Thus, it is reasonable to conjecture that Machine V will be significantly easier to program and that the cost of implementing algorithms such as floating point arithmetic and parsing functions[15] (e.g., table search), in microcode will be considerably less for Machine V than for machines such as the Intel 3002, AMP, or Burroughs Interpreter.

A. Structure



```
B.  Schematic Microcode (see Table VII for coding conventions)

    Address        Microinstruction

       1.    (STEP, NOP)/ADR(f1) =:  ALTINSAR
       2.    If p1 then (CALL, NOP) else (SKIP)/ADR(g1) =: ALTINSAR
       3.    (SKIP, NOP)/_____
       4.    (CALL, NOP)/_____

    ⎧ 10.    f1:  (STEP, NOP)/ADR(m1) =:  ALTINSAR
    ⎪ 11.    If p2 then (CALL, NOP) else (SKIP)/ADR(n1) =:  ALTINSAR
  f ⎨ 12.    (RTN, NOP)/_____
    ⎪ 13.    (CALL, NOP)/____
    ⎩ 14.    (RTN, NOP)/____

  g { 20.    g1:  (STEP, NOP)/g1
       :
      29.         (RTN, NOP)/gm

  m { 30.    m1:  (STEP, NOP)/m1
       :
      39          (RTN, NOP)/mn

  n { 40.    n1:  (STEP, NOP)/n1
       :
      49          (RTN, NOP)/nn

Note:  f, g, m, and n are all modular.
```

Figure 3—Implementation of nested if-then-else structures in modular microcode (Machine V)

SUMMARY

Current advances in semiconductor technology have led to microprogrammed and user-microprogrammable processors having a variety of microinstruction sequencing capabilities. At the present time, the primary use of microprograms is as an alternative to hardwired control sequencers in the implementation of the control function in computers with conventional instruction sets; thus, microprograms are used to implement tasks which have a relatively simple logical structure. However, it is likely that in the future microprograms will be used to support special purpose architectures with instruction sets chosen to simplify programming of certain classes of algorithms; these microprograms will be used to implement tasks which may have a relatively complex logical structure. The success of these architectures will depend on their cost-effectiveness; this includes the cost of writing and storing microcode. Thus, it is important that the microinstruction sequencing capabilities of the underlying machine organization support the implementation of the basic constructs of the appropriate program logic using "context-free" (no embedded return addresses) modules of microcode. Review of the microinstruction sequencing capabilities of several contemporary microprogram machines (the Burroughs Interpreter,[9] the Hewlett-Packard 21MX,[10] the Argonne Microprocessor,[12] the Univac C/SP,[13] and the Intel 3001[14]) has shown that these sequencing capabilities generally fail to support modular implementation of the basic constructs of flowchartable program logic. This failure is inherent in the explicit sequencing strategies implemented in machines, such as the Intel 3000 series, in which the address of the next microinstruction is obtained explicitly from the address of the current microinstruction modified by the state of the machine. However, this failure can be overcome in machines in which the address of the next microinstruction is specified implicitly by designing an appropriate set of sequencing functions. One such set of microinstruction sequencing functions (Machine V) designed to simplify microprogramming by permitting a completely modular implementation of the basic control constructs of flowchartable program logic has been described.

ACKNOWLEDGMENT

The help of David Hawk in designing Machine V and the constructive comments of the referees are gratefully acknowledged.

REFERENCES

1. Thomas, R. T., "The Development of User Microprogramming: A Survey and Status Report," *Seventh Annual Workshop on Microprogramming*, Palo Alto, October 1974 (Preprints), pp. 212-216.
2. Weber, H., "A Microprogrammed Implementation of EULER on IBM System/360 Model 30," *Comm. ACM*, 10, 1967, pp. 549-558.
3. Balzer, R. M., "An Overview of the ISPL Computer System Design," *Comm. ACM*, 16, 1973, pp. 117-122.
4. Dijkstra, E. W., "Structured Programming," in *Software Engineering Techniques*, NATO Science Committee (Eds. Burton, J. N. and B. Randell), 1969, pp. 88-93.
5. Mills, H. D., *Mathematical Foundations for Structured Programming*, Report No. FSC 72-6012, Federal Systems Division, IBM Corporation, Gaithersburg, Maryland, 1972, 62 p.
6. Hill, F. J., and G. R. Peterson, *Digital Systems: Hardware Organization and Design*, John Wiley and Sons, Inc., New York, 1973, 481 p.
7. Jones, L. H., et al., "*An Annotated Bibliography on Microprogramming I, II, and III*" (late 1969-early 1974), *SIGMICRO Newsletter*, S, 2, July 1972, pp. 39-55, 3, 2, July 1973, pp. 7-18, and 5, 2, July 1974 pp. 7-18.
8. Tucker, S. G., "Microprogram Control for System/360," *IBM Syst. J.*, 6, 1967, pp. 549-558.
9. Reigel, E. W., V. Faber, and D. A. Fisher, "The Interpreter—A Microprogrammable Building Block System," in 1972 Spring Joint Comput. Conf., *AFIPS Conf. Proc.*, 36, AFIPS Press, Montvale, N.J., 1972, pp. 705-723.
10. *Microprogramming 21MX Computers, Operating and Reference Manual*, Hewlett-Packard Co., Document 02108-90008, August, 1974.
11. Roberts, J. D., Jr., J. Ihnat, W. R. Smith, Jr., "Microprogrammed Control Unit (MCU) Programming Reference Manual," *SIGMICRO Newsletter*, 3, 3, October 1972, pp. 18-57.
12. Barr, R. G., J. A. Becker, W. P. Lidinsky, and U. V. Tantillo, "A Research Oriented Dynamic Microprocessor," *IEEE Trans. Comp., C-22*, 1973, pp. 976-985.
13. Redfield, S. R., "A Study in Microprogrammed Processors: A Medium Sized Microprogrammed Processor," *IEEE Trans. Comp. C-20*, 1971, pp. 743-750.
14. Intel 3001 Microprogram Control Unit, Product Information Bulletin.
15. De Remer, F. L., "Simple LR(k) Grammars," *Comm. ACM*, 14 1971, pp. 453-460.

Microcomputer software design—A checkpoint

by GARY A. KILDALL
Naval Postgraduate School
Monterey, California

INTRODUCTION

The general availability of low cost microcomputers has revolutionized digital design and digital applications. Using LSI chip technology, microcomputers are no more than scaled-down central processing units with minicomputer capability, and are treated as component computers at the heart of a digital design. Thus, microcomputers find wide application in both dedicated and general purpose roles, ranging from simple controllers through smart terminals and test instruments to small business data processing systems.

In each application, hardware and software modules are intermixed to minimize unit cost. As a result, the overall quality of a microcomputer-based product is directly determined by the quality of its hardware and software components. Similar to its hardware counterparts, the product's programmed subsystems must be well specified and engineered for long term reliability. In fact, well-engineered software has never been as important: packaged systems are often produced in the hundreds or thousands, where each program is permanently stored in unalterable ROM (Read-Only-Memory). Unreliable programs have far-reaching effects, while ill-specified software hinders product adaptability.

A particular high level language has emerged as an aid to the microcomputer software engineer which forecasts some industry standardization. This paper briefly reviews current design aids, with particular emphasis on applicability of high level languages in the microcomputer environment. A particular project case study is presented which exemplifies current design methodology, followed by projected trends in microcomputer software aids.

BEYOND THE DATA SHEET

In essence, a microcomputer is simply another integrated circuit chip set, with somewhat more than average capability. In fact, many design engineers consider a microcomputer CPU as simply a ROM-driven LSI chip which, with proper arrangement of 1's and 0's in the external ROM, can be tailored to act like a custom chip. The design engineer breadboards a circuit including the microcomputer, fills the ROM's with binary codes which drive the chip, and proceeds to debug with logic probe and scope. Although costly in development and maintenance time, this approach is quite popular since no external support is required beyond the chip's data sheet.

At the opposite end of the applications spectrum, the microcomputer is considered just another processor which, independent of physical characteristics, provides a key to product update and new marketing areas. Often from a minicomputer background, customers are unwilling to return to primitive programming tools and meager design support.

As a result of demands from a broad customer base, many of today's semiconductor houses find themselves in the software business. A recent survey cross-references ten microcomputer manufacturers by the software design aids which they support.[1] Of these manufacturers:

all ten support a cross-assembler,
four offer resident assemblers,
three provide a resident editor,
eight support relocatable or absolute loaders,
five provide primitive debugging facilities,
six offer cross-simulators, and
two support a high level language.

The cross products all require a larger host computer for actual execution. That is, cross-assemblers are usually written in ANSI standard FORTRAN to allow some measure of machine independence. The customer either purchases the program directly from the manufacturer, or contracts with a timesharing service which supports the manufacturer's software.

Resident software systems, on the other hand, execute using microcomputer developmental hardware. Most manufacturers offer a built-up microcomputer prototyping system as a hardware developmental aid, including CPU, memory, I/O access, and front panel control. In this configuration, the microcomputer has minicomputer characteristics, and thus can support its own software systems, including assemblers, paper tape editors, loaders, and debuggers. Although some of these resident software tools are quite comprehensive, current manufacturer's offerings are hindered by limited I/O facilities. As a result, resident software tools are less convenient than cross systems, but are generally less expensive to support.

Although similar in capability to a minicomputer, developmental systems generally incorporate features peculiar to microcomputer systems development. National's IMP-16P prototyping machine, for example, contains spe-

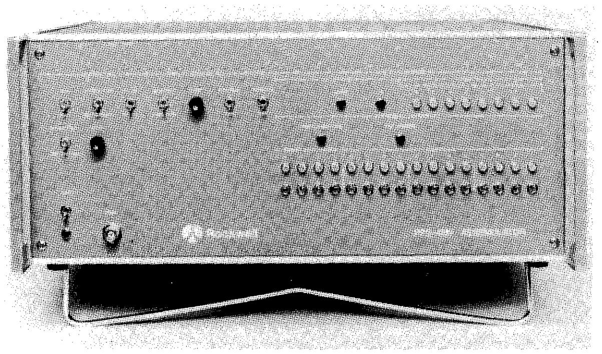


Figure 1—Rockwell's PPS-4 microcomputer development system

cial circuitry for loading reprogrammable ROM's, while Rockwell's "assemulator," shown in Figure 1, contains a built-in assembler and CPU emulator for programming and debugging their PPS-4 microcomputer. Thus, the manufacturer's developmental systems are generally inappropriate as end-user products.

Cross simulators are also used on larger host computers to programmatically simulate actions of the microcomputer. The primary problem, however, is that extensive program testing and simulation of real-time external events, such as signals input from a device controller, is tedious and expensive. Thus, cross simulators are principally used to step-through subroutines and program modules independent of the electronic environment. A simulator is extremely useful, however, when exact execution time must be determined for time-critical program segments.

Two major manufacturers of microcomputer chip sets are currently supporting a particular subset of PL/I as a base language for their products. Intel's language, called PL/M, has been available since mid-1973 through a cross-compiler, while National's product, called PL/M+, will be available in mid-1975 as an integral part of their resident developmental system. Intel's PL/M provides a base language for their 8-bit processors, and National's PL/M+ is designed for the IMP-16 and PACE microcomputers. The two languages are basically compatible, thus allowing transportation of customer software between these two manufacturers.

SYSTEMS LANGUAGES

As interest grows in PL/M-like languages for microcomputer systems development, one immediately questions the suitability of high-level languages in such an environment. First, does a language such as PL/M support necessary low-level control functions which occur in microcomputer systems, or does the designer "lose control" of his machine? Second, how memory-efficient can a translator for such a language be? The cost of high-quantity electronics products is largely determined by component count, and high-level language translators are notorious for their inefficient code sequences, resulting in excessive memory requirements in the final product. Thus, the discussion focuses on experiences with Intel's product as a benchmark for this class of languages.

First, a few general comments on PL/M itself. The language is modest in structure and scope: basic operators are tied closely to the capabilities of 8- and 16-bit processors, augmented by structures for writing assignments, simple expressions, conditional statements, looping control, and subroutine mechanisms. The result is a language which simplifies the expression of microcomputer systems, while allowing access to all machine functions, without becoming completely dependent upon a particular CPU organization. The language has facilities which are reflected within the capability of the microcomputer, and, similarly, each machine function is reflected in some high-level statement. Architecture-oriented languages of this sort, often referred to as systems languages, are traditionally used to implement the lowest level system functions to avoid the rigidities of assembly language coding. In the larger computer environment, systems languages are often used to implement operating systems, language processors, utilities, and some applications software. Thus, they are themselves self-supporting, generally requiring little existing system support. As illustrated in the examples which follow, this close relationship between the language and the machine architecture holds also for PL/M.

The Appendix contains a sample PL/M program which indicates the basic facilities of the language. This particular language has global characteristics of the "PL-family," but derives its basic structure from the microcomputer problem environment, as described above.[2]

As a final comment, one notices that after decades of ad hoc programming, there is finally an emerging body of theory and practice concerning software engineering[3,4,5,6] which is gaining industrial acceptance. Languages such as PL/M, which provide clear representation of control flows, are important tools in support of structured programming techniques. When combined with professional project management and programming practices, the result is usually well-specified, reliable, and efficient software systems.[7,8,9]

A CASE STUDY

Given the current level of support, how does one approach a microcomputer project which involves a total system design? Non-trivial projects are generally evolutionary in nature, where each phase of development and testing is a controlled experiment. In the case of software generation, the designer starts with cross systems for initial program development and testing, gradually moving to resident developmental systems, and then to a breadboarded prototype. Since system malfunctions can occur at any level, from low voltage power supplies through marginal IC's to programming blunders, this evolutionary approach isolates the range of errors at each stage. A particular microcomputer project is outlined below which demonstrates this approach.

A dedicated computer system was recently constructed at the Naval Postgraduate School to be used by Navy divers while working underwater for extended periods. The device monitors the dive time and depth, and produces a continuous read-out of the "safe ascent depth." The safe ascent depth is the depth to which the diver can ascend from his current depth without contracting the "bends." As the diver descends, his blood takes on nitrogen, and as he ascends, the nitrogen is given off. Depending upon the length of time he has worked at various depths on a particular dive, he can rise only to the safe ascent depth before nitrogen gases form in the blood. Thus, the computer keeps the diver informed of this depth. The diving computer has four principal functions to perform:

compute partial pressures of nitrogen for several controlling tissues,
monitor external parameters such as elapsed time and current dive depth,
drive simple displays with the current and safe ascent depths, and
control the sequencing of external monitoring, computing, and display.

The final prototype was developed in two man-months, with approximately three weeks devoted to software development, and the remainder in hardware design and debugging.

With the overall analysis of the dive problem complete, a BASIC program was written which computed test values. The computations involved 32-bit signed integer values with fixed precision. Since the 8 bit processors support only simple operations on 8-bit quantities, subroutines were written in PL/M to provide necessary functions. Each subroutine was compiled using the PL/M cross-compiler on the school's IBM S/360, and the machine code was read-in by another program, called INTERP/8, which simulates 8008 CPU actions. Using the break point and display commands of the simulator, the numeric subroutine package was checked-out, using only the S/360, with no physical microcomputer hardware.

The numeric subroutines were augmented by additional PL/M coding which evaluated standard formulae (essentially the same as those of the BASIC program) for determining the partial pressures of nitrogen for a particular depth. Again, these subroutines were checked-out under simulation by inserting test values in simulated memory, running a single computation, and displaying the values resulting from the simulation. A control and sequencing program was then written which simulated a complete dive by looping through a predetermined dive profile of times and depths. Using the simulation, several complete dive profiles were run, and the intermediate and final results were compared with the BASIC program. Extensive testing was infeasible, however, since a simulated fifteen minute dive to a depth of 130 feet required over thirty minutes of S/360 CPU time.

Transition to real microcomputer hardware thus became necessary to complete the testing. From this point on, the program was compiled using the cross PL/M compiler on the S/360, but executed in real time using a developmental system. A paper tape was produced from the S/360 compilation containing the 8008 machine code which was then loaded through the Teletype reader into the memory of the developmental system, and executed.

In order to properly check-out the central algorithms, another set of subroutines was written in PL/M which provided basic communication between the program and Teletype, allowing the program to read commands, write test results, and read and print 32-bit fixed point numbers. These subroutines formed a software test bed which would eventually be discarded. Each test involved a dive profile with various times and depths preset from the Teletype console. The program would run the dive profile and print the safe ascent depth at crucial points in the test. The computations executed in five times real time (a 30 minute dive was completed in six minutes of 8008 time), and thus it was possible to verify results by comparing with both the BASIC program and standard Navy diving tables. After check-out, the central algorithms were separated from the test environment, and set aside for the final prototype.

At this point, it was determined that there were several disadvantages in using the 8008 for the final prototype, including factors such as power consumption and compactness. Thus, the design was altered to incorporate the newer 8080 microcomputer. Because of its increased speed, the 8080 could be "shut-down" for longer periods between each computation, resulting in significant power savings (partial pressures were updated every two seconds, and could be computed in 50 milliseconds). The PL/M language is upward compatible along this processor line, and thus the program was recompiled using the 8080 version of PL/M.

The prototype was constructed and debugged, and, upon completion, I/O drivers were coded in PL/M, placed into erasable ROM in the prototype, and independently tested. The I/O drivers were then combined with the core computation and control algorithms. The total program was compiled on the S/360, placed into ROM in the prototype and checked-out. As shown in Figure 2, the completed prototype is contained on a single 7×9 wirewrap board with space for 2K bytes of erasable ROM (the program currently uses 1.2K), and 1024 bytes of random access memory.

ADDITIONAL APPLICATIONS

The case study given above serves to illustrate current methods used to develop dedicated microcomputer software. In addition, the application involves both bit-level and simple numeric processing, which are both handled well in this particular high level language. To

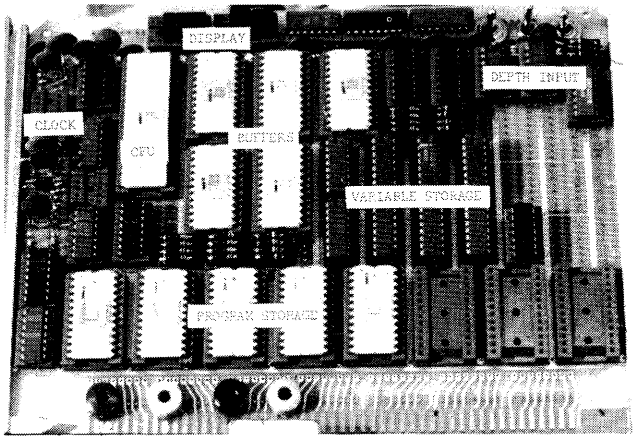


Figure 2—Navy SCUBA diving computer, using the Intel microcomputer

illustrate the range of applicability of PL/M, however, additional projects from more traditional computer areas are considered.

There is current industry-wide interest in incorporating today's low-cost peripherals with microcomputer devices to build inexpensive general purpose processors for resident microcomputer development and end-user applications. One such computer system, shown in Figure 3, includes a floppy disk operating system, which implements a named file structure with dynamic disk allocation on multiple disks, sequential or random access, and optimal disk arrangement strategies. When combined with the system's loaders, language processors, editors, and debuggers, the resulting facility rivals that of most time-sharing services for microcomputer program development. All software modules are written in PL/M including basic file management subroutines (3K), transient console command handler (2K), and various utility programs. An indefinite number of programs and subsystems can be supported since they reside on disk, and are loaded into memory on demand. Clearly, this particular application of a microcomputer heavily overlaps traditional general-purpose minicomputer areas.

A number of language processors have been implemented in PL/M, including a translator for the BASIC language as an aid in developing microcomputer programs which make heavy use of floating point operations. The BASIC translator operates under the disk system described above, and produces code which is executed interpretively by a special run-time subroutine package. More importantly, any translated program can optionally be loaded into ROM with the run-time subroutines, and placed into a circuit with a microcomputer which executes the program repetitively at the push of a button.

The translator for BASIC was itself written in PL/M (5K), and demonstrates its use as an implementation language. That is, PL/M has only simple operations, and thus is relatively easy to implement for any microcomputer. Given that PL/M exists, further special-purpose programs, such as the BASIC translator can be coded easily. As a result, all system software can be transported between different architectures if the base language can be transported. It is reassuring to know, for example, that the disk system software, BASIC translator, and BASIC programs will execute on Intel's 8008 and 8080 machines, as well as National's IMP-16 and PACE microcomputers with little modification.

SUITABILITY OF PL/M

These examples indicate the suitability of one high-level language in microcomputer systems design. Based upon this implementation, the most straightforward applications were those which the basic machine could already perform, including bit-level I/O control and character manipulation found in word-processing, operating systems, and language processors. In these cases, the algorithms were easy to express, and simple to debug and maintain. The operating system application, however, contains heavier use of table subscripting and run-time address computations. Although these functions were easy to express in PL/M, the underlying computations are more complicated for Intel's 8-bit machines. General floating point applications were by far the most complicated to code and debug in PL/M and, in general, resulted in a sequence of unintelligible mainline calls on these numeric subroutines.

The question of memory-efficiency is also a part of the suitability discussion. Again, the bit-level and character processing functions result in short code sequences which are quite competitive with good assembly language programming. The 16-bit address computations found in operating system work cause excessive program length unless the programmer uses techniques, such as localizing computations to common subroutines, which minimize this overhead. The general floating point application took an inordinate amount of program storage, due principally to the lack of basic machine facilities to perform these functions. One should consider implementing basic arithmetic functions of this sort in PL/M-compatible

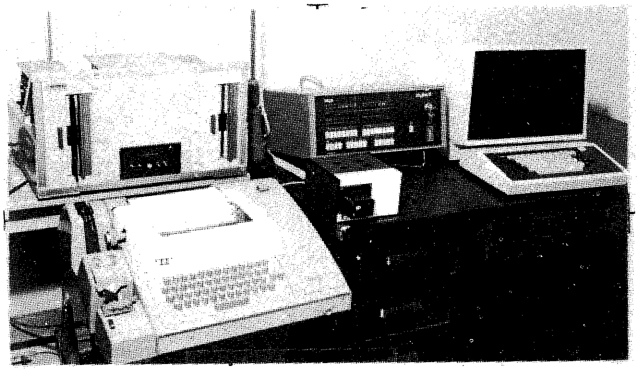


Figure 3—A disk-based microcomputer development system

assembly language where the side-effects of the machine can be more easily exploited. In any case, measured overhead for PL/M is in the range 10 percent to 35 percent when compared with assembly language coding, based upon experienced programmers and the current PL/M compiler.[9]

One can conclude, however, that the most suitable problems for expression in PL/M are precisely those problems which are most appropriate for the 8-bit processors. That is, the low-level functions are all present in PL/M, and the high-level functions are not. Further, the low-level functions are exactly the ones which are most memory-efficient.

FUTURE TRENDS

Microcomputer development practices seem to change on a monthly basis as manufacturer support increases, and hardware component costs decrease. Although any projections are questionable in light of this advancing technology, several trends are evident. First, the use of inconvenient and expensive cross development tools will be short-lived. Although the cost for cross assembly and cross compilation is comparable, either approach can rapidly consume project funds. Inexpensive disk-based resident developmental machines are becoming commercially available which, although still somewhat primitive, can be purchased for the price of the timesharing services necessary for even a moderate project. National's PL/M+, for example, will be available in mid-1975 as an integral part of their floppy disk-based development system, while numerous independent companies are providing add-on equipment for Intel, Rockwell, and other manufacturers. Due to the developmental nature of these systems, resident language processors will soon be augmented by comprehensive debuggers which provide high level reference through symbolic names and statement context.

Current interest in PL/M as a base language indicates that high level language standards are possible to some degree in the 8-bit processor category. Although there are obvious customer benefits in training, documentation, benchmarking, program portability, and machine independence, standardization also benefits the manufacturer. The present similarity between Intel's PL/M and National's PL/M+ allows the companies to "second source" one another at the language compatibility level. Thus able to share customer bases, their products can compete on a meaningful level: questions of suitability are settled by benchmarked performance and cost, not simply on the cycle time of the CPU. The role of the microcomputer has expanded since the initial introduction of PL/M, however, and thus the language must evolve to suit these applications. Nearly all major manufacturers have investigated the implementation of a PL/M-like language for their processors, and one can only guess whether these factors will lead to a unified base language, or simply a maze of confused dialects.

REFERENCES

1. Falk, H., "Microcomputer Software Makes its Debut," *IEEE Spectrum,* Vol. 10, No. 11, October, 1974.
2. *A Guide to PL/M Programming,* Intel Corporation, 3065 Bowers Ave., Santa Clara, Ca., 95051.
3. Buxton, J., *Software Engineering Techniques,* Nato Science Committee, OTAN/NATO, 1110 Bruxelles, Belguim, April, 1970.
4. Dahl, et al., *Structured Programming,* Academic Press, 1972.
5. Kernighan, B., et al., *The Elements of Programming Style,* McGraw Hill, 1974.
6. Yourdon, *Advanced Programming Techniques Volume 1: Program Structure and Design,* Yourdon, Inc., New York, N.Y., 1974.
7. Davidow, W., "Processors and Profits: How Microprocessors Boost Them," *Electronics,* July 11, 1974.
8. Metzger, *Managing a Programming Project,* Prentice Hall, 1973.
9. Kildall, G., "Systems Languages: Management's Key to Controlled Software Evolution," *Proceedings of the 1974 Western Electronics Show and Convention,* September, 1974.

APPENDIX

The listing given in Figure 4 is an example of an 8080 PL/M program which executes on an Intel developmental system. The purpose of the program is to test a procedure which keeps track of the elapsed time since system start-up. After each minute of elapsed time, the program prints:

hh HRS mm MINS

at the teletype, where hh and mm are decimal values for the hours and minutes of elapsed time.

The following run-time environment is assumed. A Teletype is connected to the 8080 CPU through a UART (Universal Asynchronous Receiver-Transmitter). In addition, an external interrupt is generated every 1/60th of a second, and is used for the basic program timing.

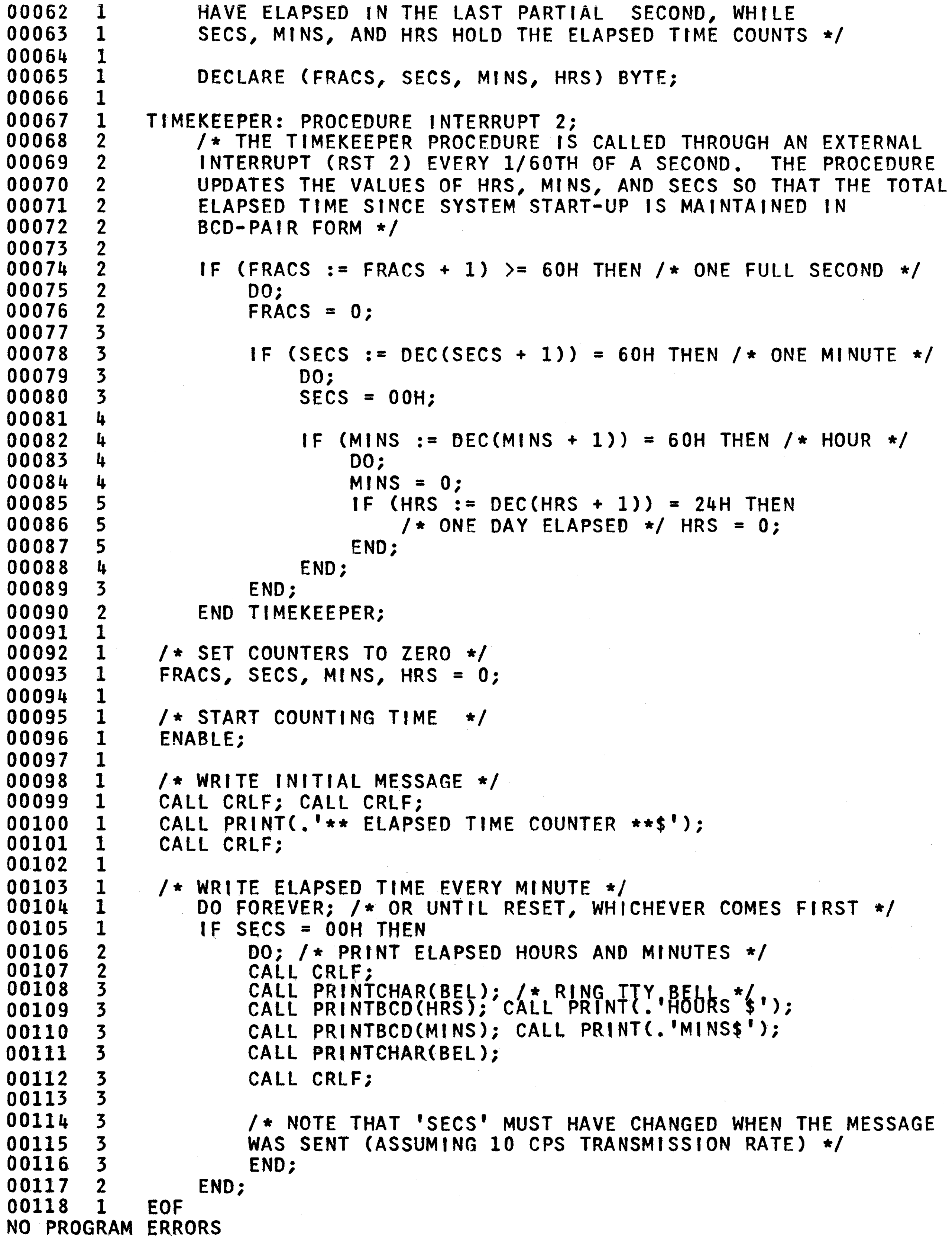
The program consists of a number of procedures followed by calls on these procedures. The mainline procedures are listed below along with their function in the program:

PRINTCHAR	print the single ASCII character in CHAR
CRLF	send a carriage-return and line-feed
PRINTBCD	print two decimal digits
PRINT	print a sequence of characters

One "interrupt procedure," called TIMEKEEPER, is defined with the attribute INTERRUPT 2. This interrupt attribute results in control transfer to TIMEKEEPER whenever interrupts are enabled and the external interrupt occurs.

The first PL/M statement which is executed follows the TIMEKEEPER procedure. The four variables FRACS, SECS, MINS, and HRS are zeroed. The first variable, FRACS, is a byte variable which tallies the number of 1/60ths of a second which have elapsed during a one second interval. The remaining variables each hold a pair of BCD

```
1     /* THE FOLLOWING 8080 PL/M PROGRAM COMPUTES AND DISPLAYS THE
1        ELAPSED TIME SINCE SYSTEM START-UP.  THE ELAPSED TIME IS
1        PRINTED AT THE TELETYPE CONSOLE EVERY MINUTE */
1
1    DECLARE
1        /* LITERAL SUBSTITUTIONS IN THE PROGRAM */
1        TRUE LITERALLY '1',
1        FALSE LITERALLY '0',
1        FOREVER LITERALLY 'WHILE TRUE',
1
1        /* TELETYPE CONSTANTS FOR UART */
1        TTO LITERALLY '0',   /* DATA TO TTY IS OUTPUT(0) */
1        TTS LITERALLY '1',   /* STATUS PORT IS INPUT(1) */
1
1        /* SPECIAL CHARACTERS (NON GRAPHIC) */
1        BEL LITERALLY '7',   /* RING TELETYPE BELL */
1        CR LITERALLY '15Q',  /* CARRIAGE RETURN (15 OCTAL) */
1        LF LITERALLY '0AH';  /* LINE FEED (A HEXADECIMAL) */
1
1        /* TELETYPE OUTPUT SUBROUTINES */
1
1    PRINTCHAR: PROCEDURE(CHAR);
2        DECLARE CHAR BYTE;
2        /* PRINT THE 8-BIT ASCII CHARACTER IN 'CHAR' AT THE
2        TELETYPE CONSOLE */
2
2            DO WHILE ROR(INPUT(TTS),2);
2            /* WAIT FOR UART TRANSMIT READY */
2            END;
2
2        OUTPUT(TTO) = NOT CHAR;
2        END PRINTCHAR;
1
1    CRLF: PROCEDURE;
2        /* SEND A CARRIAGE-RETURN FOLLOWED BY A LINE-FEED */
2        CALL PRINTCHAR(CR); CALL PRINTCHAR(LF);
2        END CRLF;
1
1    PRINTBCD: PROCEDURE(B);
2        /* PRINT THE BCD-PAIR HELD IN THE 8-BIT VARIABLE 'B' */
2        DECLARE B BYTE;
2        CALL PRINTCHAR(SHR(B,4) + '0');
2        CALL PRINTCHAR((B AND 0FH) + '0');
2        END PRINTBCD;
1
1    PRINT: PROCEDURE(A);
2        /* WRITE CHARACTERS TO THE TELETYPE STARTING AT ADDRESS 'A'
2        IN MEMORY UNTIL THE FIRST '$' CHARACTER IS ENCOUNTERED */
2        DECLARE A ADDRESS,
2            (MESSAGE BASED A) BYTE;
2
2            DO WHILE MESSAGE <> '$';
2            CALL PRINTCHAR(MESSAGE);
3            A = A + 1;
3            END;
2
2        END PRINT;
1
1        /* END OF TELETYPE OUTPUT SUBROUTINES */
1
1        /* FRACS HOLDS THE NUMBER OF 1/60THS OF A SECOND WHICH
```



```
00062  1        HAVE ELAPSED IN THE LAST PARTIAL  SECOND, WHILE
00063  1        SECS, MINS, AND HRS HOLD THE ELAPSED TIME COUNTS */
00064  1
00065  1        DECLARE (FRACS, SECS, MINS, HRS) BYTE;
00066  1
00067  1    TIMEKEEPER: PROCEDURE INTERRUPT 2;
00068  2        /* THE TIMEKEEPER PROCEDURE IS CALLED THROUGH AN EXTERNAL
00069  2        INTERRUPT (RST 2) EVERY 1/60TH OF A SECOND.  THE PROCEDURE
00070  2        UPDATES THE VALUES OF HRS, MINS, AND SECS SO THAT THE TOTAL
00071  2        ELAPSED TIME SINCE SYSTEM START-UP IS MAINTAINED IN
00072  2        BCD-PAIR FORM */
00073  2
00074  2        IF (FRACS := FRACS + 1) >= 60H THEN /* ONE FULL SECOND */
00075  2            DO;
00076  2            FRACS = 0;
00077  3
00078  3            IF (SECS := DEC(SECS + 1)) = 60H THEN /* ONE MINUTE */
00079  3                DO;
00080  3                SECS = 00H;
00081  4
00082  4                IF (MINS := DEC(MINS + 1)) = 60H THEN /* HOUR */
00083  4                    DO;
00084  4                    MINS = 0;
00085  5                    IF (HRS := DEC(HRS + 1)) = 24H THEN
00086  5                        /* ONE DAY ELAPSED */ HRS = 0;
00087  5                    END;
00088  4                END;
00089  3            END;
00090  2        END TIMEKEEPER;
00091  1
00092  1     /* SET COUNTERS TO ZERO */
00093  1     FRACS, SECS, MINS, HRS = 0;
00094  1
00095  1     /* START COUNTING TIME  */
00096  1     ENABLE;
00097  1
00098  1     /* WRITE INITIAL MESSAGE */
00099  1     CALL CRLF; CALL CRLF;
00100  1     CALL PRINT(.'** ELAPSED TIME COUNTER **$');
00101  1     CALL CRLF;
00102  1
00103  1     /* WRITE ELAPSED TIME EVERY MINUTE */
00104  1        DO FOREVER; /* OR UNTIL RESET, WHICHEVER COMES FIRST */
00105  1        IF SECS = 00H THEN
00106  2            DO; /* PRINT ELAPSED HOURS AND MINUTES */
00107  2            CALL CRLF;
00108  3            CALL PRINTCHAR(BEL); /* RING TTY BELL */
00109  3            CALL PRINTBCD(HRS); CALL PRINT(.'HOURS $');
00110  3            CALL PRINTBCD(MINS); CALL PRINT(.'MINS$');
00111  3            CALL PRINTCHAR(BEL);
00112  3            CALL CRLF;
00113  3
00114  3            /* NOTE THAT 'SECS' MUST HAVE CHANGED WHEN THE MESSAGE
00115  3            WAS SENT (ASSUMING 10 CPS TRANSMISSION RATE) */
00116  3            END;
00117  2        END;
00118  1    EOF
NO PROGRAM ERRORS
```

Figure 4—A sample PL/M program for the 8080 microcomputer

numbers. The ENABLE statement turns on the 8080 interrupt system.

At this point, the program execution must be considered in two parts: the mainline code which continues past the ENABLE statement, and the interrupt code which is executed each time an interrupt is generated. If the interrupt system had not been enabled, the mainline code within the DO FOREVER block would execute indefinitely, and, since the value of SECS remains at zero, the message

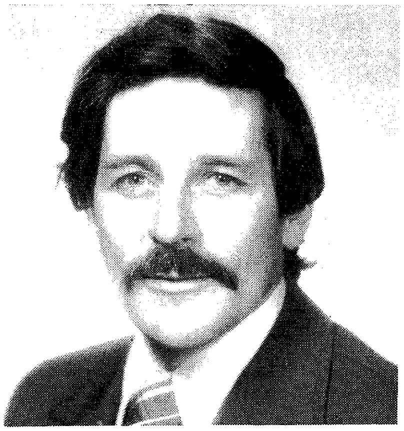
```
00 HRS 00 MINS
```

would print continuously.

Given that the interrupt system has been enabled, the interrupt which occurs 60 times each second causes the mainline code to stop at each interrupt. The TIMEKEEPER procedure immediately receives control, with the interrupt system automatically disabled and the machine state saved. Upon completion of the interrupt processing, control returns back to the interrupted mainline code to the point of interruption with the machine state restored, and interrupts enabled. As a result, the values of SECS, MINS, and HRS are continuously incremented as the mainline program executes. Thus, the program output will appear as follows:

```
00 HRS 01 MINS
00 HRS 02 MINS
00 HRS 03 MINS
```

and so-forth, with one minute intervals between each line of output.



Area Director:
Robert F. Daly
Stanford Research Institute
Menlo Park, California

Computer communications networks

Advances in computer and communication technologies are profoundly affecting the nature, structure, and operation of computer-communications networks. The joint exploitation of these technologies while promising seemingly unlimited operational possibilities, generates an almost endless parade of issues for the designer, supplier, user, and regulator of such networks. This series of four sessions reports on the results of recent research on various technical issues in the area of computer-communications networks.

An introductory session has been organized to prepare the novice for the later more technical sessions and to inform the expert of current status. This session is followed by three state-of-the-art sessions on substantive research activities affecting current and future operational systems.

A session on advances in packet switching includes papers on system design considerations and issues, speech transmission in packet networks, and control schemes for multi-access broadcast packet channels. A session on the future impact of packet radio addresses issues in the design of packet radio networks, the technology of packet radio and personal terminals. Finally, a session on advanced data communications in computer-communications networks will be included.

Computer communications—How we got where we are

by IVAN T. FRISCH and HOWARD FRANK
Network Analysis Corporation
Glen Cove, New York

HENRY FORD WAS WRONG

Of course, Henry Ford was wrong. History is not bunk. It just tends to look like bunk in the short range. Legitimately, historians must allow some time for the confusion of events to die away. They can then evolve theories about fading memories of the events. In his short story, "The Ugly Little Boy," Isaac Asimov[1] has a reporter say the following about a machine which recovers people from the past and makes them live in the present. "You can only reach out so far; that seems sensible; things get dimmer the further you go; it takes more energy. But then, you can only reach out so near." It is the same with history.

Accordingly, some of the giants of history still have only little to say about the computer revolution. Arnold Toynbee, who has provided the greatest conceptual unification of world history in this century, is still involved in the purely negative aspects of the revolution. His chapter on computerization[2] is called "Mechanization, Regimentation and Boredom"; this brings to mind some advice for fourteenth century magicians, "If you want to be a successful prophet, prophecy evil."[3] Daniel Boorstin, winner of the Bancroft Prize, the Parkman Prize and the Pulitzer Prize for his penetrating series of books, "The Americans," is most fascinated by the gadgetizing of Americans: "When automation became widespread and electronic computers became almost as common as the adding machine, there were new cataclysms in the jobs of Americans and in their ways of thinking. By 1967, only a half-century after the first commercially successful billing machine, the annual American production of cash registers and computing machines totaled more than $4.5 billion. When precise and up-to-date information was available about the quantities of everything, businessmen and consumers could not help thinking quantitatively."[4]

The facts are right, but the impact is trivialized. This nearsightedness, being fairly general among historians, we therefore seek for the general history and impact of computer-communications elsewhere. We must search among the participants, namely ourselves, and among other commentators, who will be broadly classified as journalists. One must be wary of Marshall McLuhan's generalizations. After all, in their book, "War and Peace in the Global Village," McLuhan and Quentin Fiore attribute the age of chivalry to the invention of the stirrup.[5] Granted that McLuhan is not a master of understatement, one can still find truth in his estimation, in the same book: "The computer is by all odds the most extraordinary of all technological clothing ever devised by man, since it is the extension of our central nervous system. Beside it, the wheel is a mere hula-hoop." One of the best journalist historians is James Martin. After all, he published a book in 1971 called, "Future Developments in Telecommunications," and much of this book intended as almost science fiction for the year 1980 is a good history of the years 1971-1974.

SEPARATING THE USER FROM HIS COMPUTERS

In 1939 Aikin and a team of IBM engineers at Harvard began the work that resulted in 1944 in the Mark I, the first automatic electromechanical digital computer. The first completely electronic computer was designed by Eckert and Mauchly at the University of Pennsylvania, for the Ballistic Research Laboratory at Aberdeen. The ENIAC (Electronic Numerical Integrator and Calculator) became operational in 1946. The history of computing in the 30 years since Mark I is a monumental one, which will require some new giants of history. For the present, we will try to simply indicate some of the trends and milestones, in the more limited area of computer-communications, or computer networking or, in simpler terms, the process of separating the user from his computer. We will subdivide this process into two categories—terminal oriented networks and the area with the shorter history, but greater technical promise, computer to computer networks.

EVOLUTION OF TERMINAL ORIENTED NETWORKS

The first computer network consisted of a computer with several cables attaching input devices. A majority of the networks in the world are still of this fashion. Very shortly, the need arose to do more than just communicate with a computer 100 feet away. Thus, remote terminals were added to the network. The networks were first extended to cover all of the buildings within an industrial

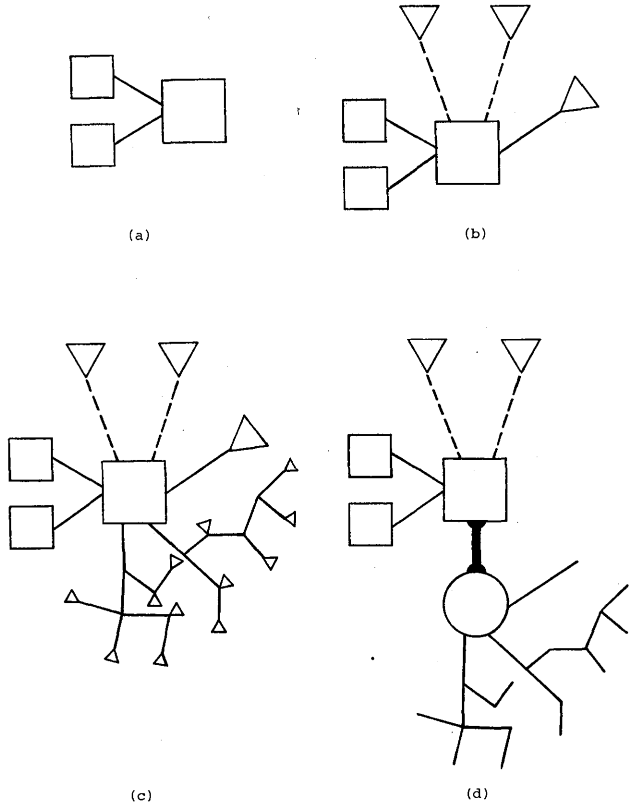


Figure 1—Evolution of terminal oriented networks

complex on leased or specially constructed lines. The capability to dial into the main frame computer was then added and the networking era began in earnest.

As networks grew, their costs also grew, often quite rapidly. For example, as more and more demands were made on the system, the cost of the *communications* became a very significant fraction of the cost of the overall network. Originally, the computer represented the majority of the total system cost. But, as the network expanded, communications often exceeded 50 percent of the overall system cost. Therefore, efforts began to reduce this aspect of the overall cost. Innovations like multidrop lines, which allowed a number of different terminals to share a common line, were introduced to take advantage of all possible economies of scale. You might be able to lease a very low bandwidth line for, let's say, a thousand or fifteen hundred dollars per month. On the other hand, you could probably increase the capacity of the line by a factor of ten or more at a cost increase of only a factor of two. This provided sufficient capacity to allow sharing of the line by several terminals. But, to do this, control mechanisms for selecting different terminals on the line and for protecting data had to be invented and techniques for contention resolution and queueing were required.

The next major difficulty encountered in building computer networks were the changes to the main frame software which were found to be exceptionally difficult and costly. Thus, to reduce the time and cost of system development, devices called "front ends" were introduced. These allowed the communication functions of the computer network to be separated, by and large, from the processing function of the computer. Front end use grew very rapidly, beginning in the late 1960's and was assisted by the introduction of low cost minicomputers. Today, front ends play an important role in network communications.

Next, the interesting observation was made that there was a cable between the front end and the computer. Since large networks always tend to get larger, the cable became longer, with communication equipment required between the front end and the computer. As the front end increased its distance from the main frame, its name changed to that of "concentrator." In modern networks, concentrators may be thousands of miles from the computer. Their main function is to reduce communication cost by more effective communication line utilization. The next development was quite natural; another front end was added to the computer side of the network to complete the isolation of the computer from its network elements.

In Figure 1, we have a typical structure of a terminal oriented network.[6] This particular network is called the NASDAQ System. "NASDAQ" stands for the National Association of Securities Dealers Automated Quotations System. This network was built in 1970 and became operational in 1971. Its function is to collect quotation information about the Over-the-Counter Securities market. Users distributed throughout the country receive responses to their input in five or six seconds. Responses contain information about the prices at which dealers are willing to sell or buy securities, and the exact bid and ask prices of each market maker who deals in a particular security. There are on the order of 1,700 terminals in this system at a thousand different locations in about 400 different cities. The system has reduced the problem of getting the information about Over-the-Counter stocks from one of making ten phone calls to the input of a single network message. During active trading days, the NASDAQ System has handled more than one million messages a day.

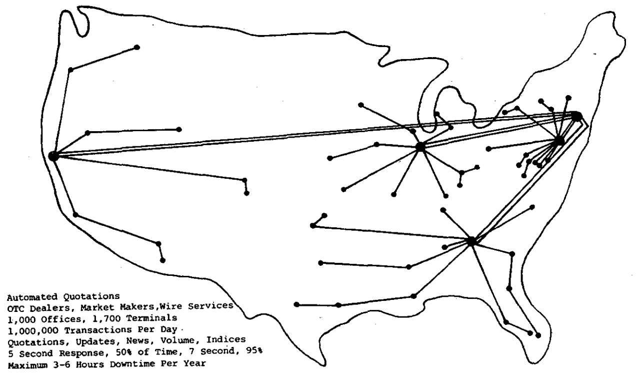

Automated Quotations
OTC Dealers, Market Makers,Wire Services
1,000 Offices, 1,700 Terminals
1,000,000 Transactions Per Day
Quotations, Updates, News, Volume, Indices
5 Second Response, 50% of Time, 7 Second, 95%
Maximum 3-6 Hours Downtime Per Year

Figure 2—Simplified network diagram for the NASDAQ system

MILESTONE TERMINAL ORIENTED NETWORKS

There are almost as many terminal oriented systems at present as there are computers, since almost every computer has terminals attached to it. And almost all these systems fit somewhere into the evolutionary pattern we have described. However, only a small number of these networks set milestones in either timing, structure, function or size. Those that have been major benchmarks fall into two general categories: special purpose networks—intended to serve a specific function for a selected set of users—and time sharing services—intended as a general utility for any user.

Special purpose networks

Military

Among special purpose networks the military has been one of the leading users and pioneers. Indeed, much of the technology developed for military purposes has been transferred and adapted for commercial use. The prime examples are point of sale systems, of which banking and airline reservation systems are pioneering areas. Other users such as educational institutions have also added major improvements necessitated by their particular requirements. Certainly a milestone in military systems and in computer communications development, in general, is the SAGE (Semiautomatic Ground Environment) system. Lest we forget in how many different ways this system was a pioneering effort I will quote Ruth Davis:

"The first use of an automated display which permitted the user to exercise control over the information presented (and also to enter requests and information based on what was presented to him) occurred in the SAGE system. The significance of the introduction into this system of the light gun as a pointing device under the control of the display operator cannot be overemphasized. It was probably the one most important event which made possible the man-computer interaction deemed so essential at the present time. It occurred in 1952 utilizing the Whirlwind computer."[7]

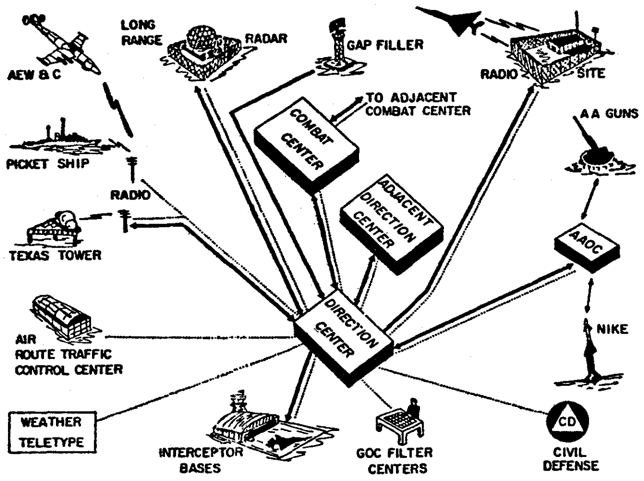


Figure 3—Typical sage sector
——— Data circuits
. . . . Other circuits

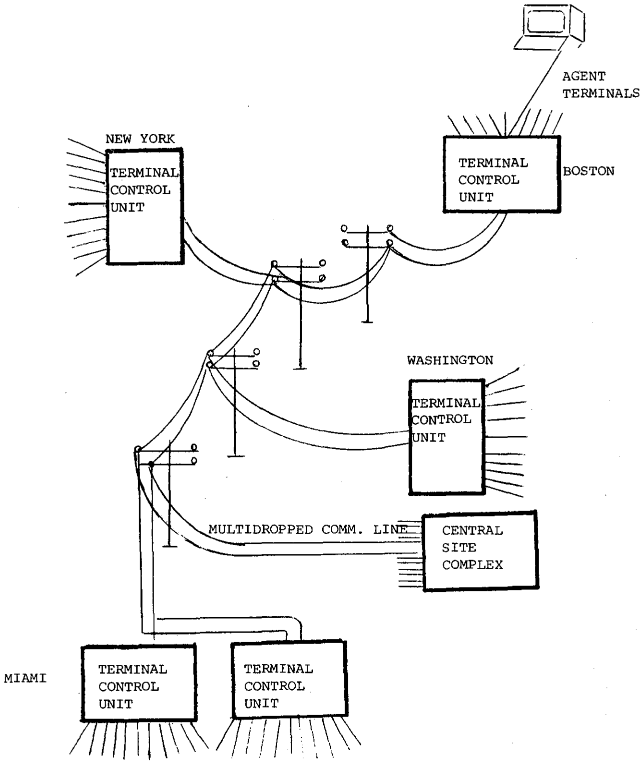


Figure 4—Airlines reservations system

But let us look at the computer communications aspects. The purpose of the system was air defense for the U.S. The results were benchmark efforts in computers, communications and computer communications.

The Air Defense System Engineering Committee (ADSEC), a group formed by the Scientific Advisory Board at the request of the Air Force, evaluated the status of overall air defense in the 1950's. They recommended initial feasibility tests utilizing digital radar inputs to a central computer. This was to be accomplished by coupling the data-processing capabilities of the Digital Computer Laboratory to the radar data-transmission techniques of the Cambridge Research Center. Favorable results led to Project Charles and the establishment of Lincoln Laboratory in 1951 with a charter to work toward a computer-based air defense system. Project Charles activities led to recommendations for a prototype test facility known as the Cape Cod System, which was established in 1952.

The New York Air Defense Sector became the first operational site in 1958. By 1963, SAGE Direction Center and Combat Centers had been installed at all continental stations. The system was designed in 1955 with IBM AN/F SQ-7 prototype computers, with SDC software at the central facilities. Each computer contained 58,000

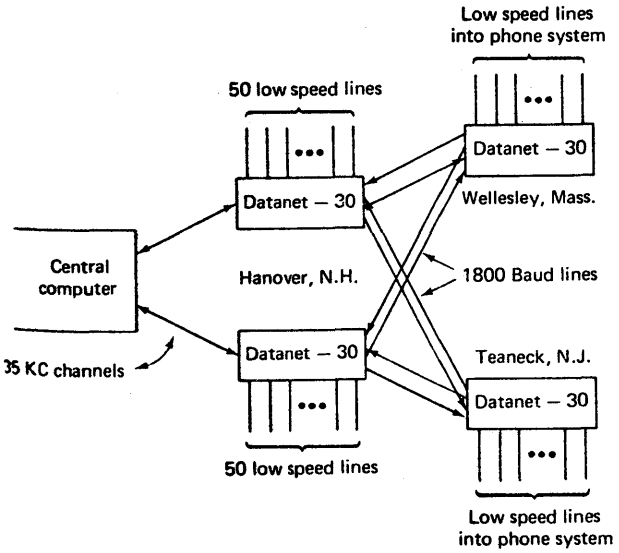


Figure 5—Dartmouth time sharing system interconnections between remote and local communications computers (1968)

vacuum tubes, consuming 1,500 KWatts of power and occupying an entire building floor.[8] Radars and information sources feed information to the centers and the centers send information to interceptors and other weapons. Real time processing required key developments by many companies, small computer (not mini computers) front-end processors, specification of 1600 baud data lines with better conditioning than voice grade lines, and redundant diverse routed paths for reliability.

Banking

The development of commercial systems such as banking could be done on a smaller scale and hence had less auspicious milestones. Certainly, the first of any system must be a milestone. The first banking milestone therefore sounds almost like an entry from the Guiness book of records. Telefile is described by Sackman[7] as the first online banking system in the world, linking the transactions of each of the three participating banks and their affiliated branches into a central data-processing system. This system grew out of automation feasibility studies initiated by the Howard Savings Insitution of Newark, New Jersey in 1953. By 1956, system requirements were specified, two other banks cooperated in the venture, and the Teleregister Corporation was awarded the contract for developing and implementing the data-processing system.

The three main system requirements were as follows:

1. Online data processing at the teller window—for example, direct communication between the teller and the central computer for deposits and withdrawals.
2. High system reliability and accuracy commensurate with rigorous banking standards.
3. Uninterrupted continuity in banking service throughout the transition period from the initial manual system to the successor semi-automated system.

The system is a long way in scope from present broad purposed vast networks such as that of the Barclay Bank or that being considered by the Federal Reserve Board.

Airlines

One of the earliest large scale users of point of sale type systems have been the airlines. As Janet Taplin[9] has commented "American Airlines has been uniquely successful in its use of computers. Its SABRE I was the first on-line reservation system and represented a major breakthrough in terms of real-time computer usage". A joint research effort by IBM and American Airlines in the early 50's culminated in the SABRE system in the early 60's. The system consists of a central computer site with 2000 nationwide terminals multidropped to the central site.[10]

Education

One of the earliest and most ambitious educational networks is the Dartmouth Time Sharing System (DTSS) first placed in operation in 1964.

"It was . . . decided that exposure to computing and free availability of computing should become a standard part of the liberal arts educations at Dartmouth, an undergraduate college where only 25 percent of the students elect majors in the sciences and engineering.

. . . Against this background, it was recognized that the user-computer interface had to be simplified and harmonized with the educational environment if liberal arts students were to ingest a reasonable dose of sensible knowledge about computing. Two important consequences of this recognition were the decisions to bring the computer to the student via remote individual terminals (teletypes) and to devise an extremely simple user interface."[11]

The system evolved through several stages of hardware and software systems as well as communications. The use of DTSS by schools outside Dartmouth developed sporadically until given a major impetus in 1967-1968 by NSF Grants. The configuration in 1968 is shown in Figure 5.

Time-sharing networks

The emergence of time-sharing systems as general purpose on line computing facilities is a development primarily of the 1960's. Some of the early experimental work took place at Project MAC at MIT; SDC under the aegis of ARPA; and RAND. By the mid 1960's practically all computer manufacturers were marketing or developing some form of time-sharing facilities. A number of organizations now run commercially available time shared services. Among them are United Computing Services, Inc., Utility Network of America and so on.

The most significant networks are unusual in function, size and complexity.

The largest time sharing network is that run by General Electric.[12,13] It has local data lines in some 25 cities in the U.S., nine cities in Canada, Mexico City, San Juan and via COMSAT, London, Manchester, Brussels, Amsterdam and Paris. The system evolved from GE's experience with the Dartmouth Time Sharing System and in 1965 used the operating system developed at Dartmouth.

The most sophisticated time-sharing networks currently in operation is TYMNET owned by Tymshare, Inc.[12,14] The network employs 80 communications processors all over the U.S. accessing 26 host computers. The network configuration consists of a backbone of multiple rings, rather than a star, with other nodes connected in stars or straight runs. If one path to a computer is saturated or down, the network automatically switches to an alternate path. The network goes far beyond the concept of individual real time terminal users and services entire organizations such as major accounting firms and the National Library of Medicine.

COMPUTER TO COMPUTER NETWORKS

Parallel to the development of terminal oriented systems, efforts were under way to allow computers to directly communicate with other computers in real time. The first step was, of course, to place two identical computers in the same building and to connect a cable between them. (Many of the computers being built today can be regarded as sophisticated computer networks in themselves.) To assist in this difficult task, devices very much like front ends were developed to handle the com-

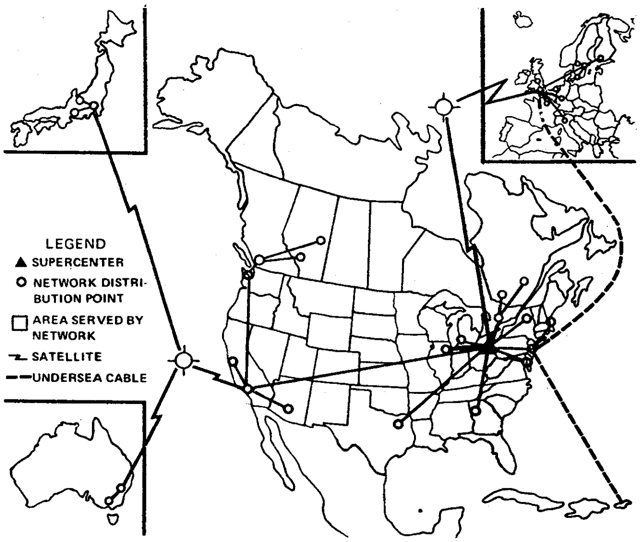


Figure 6—General Electric international network

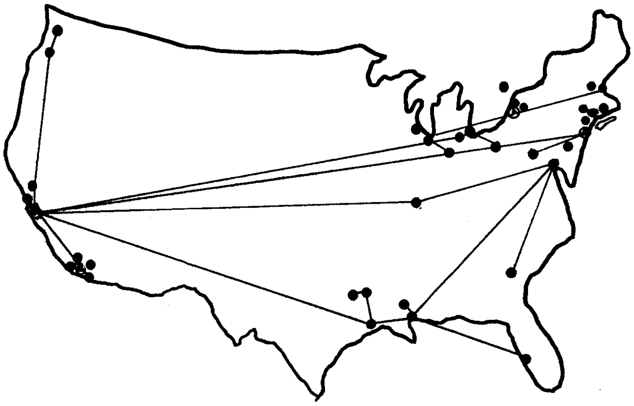


Figure 7—Tymnet

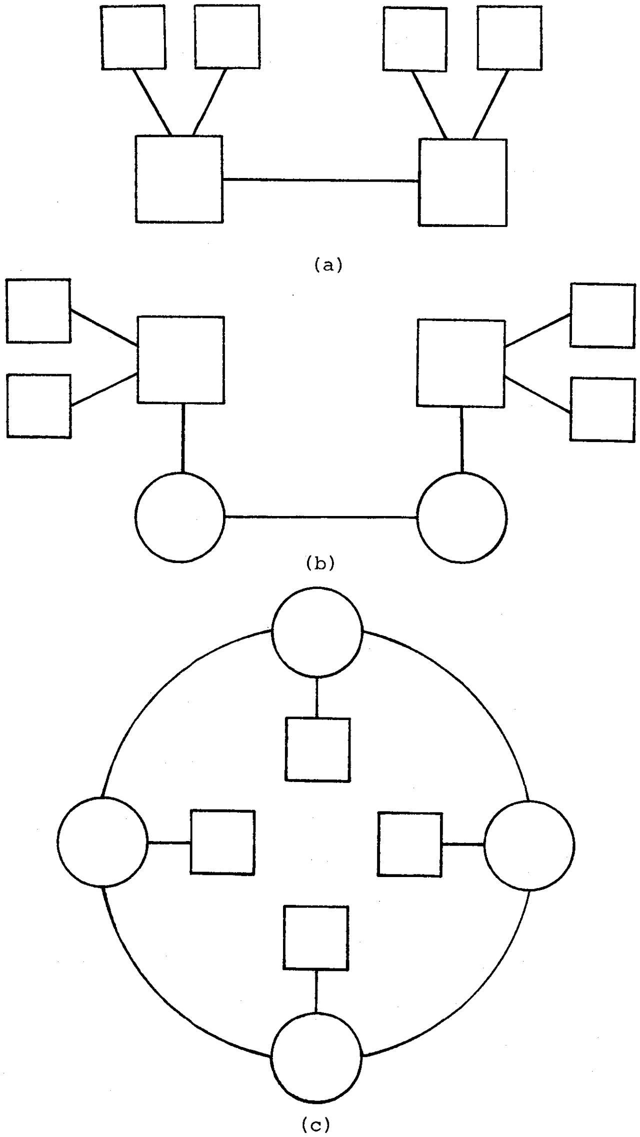


Figure 8—Evolution of computer-to-computer networks

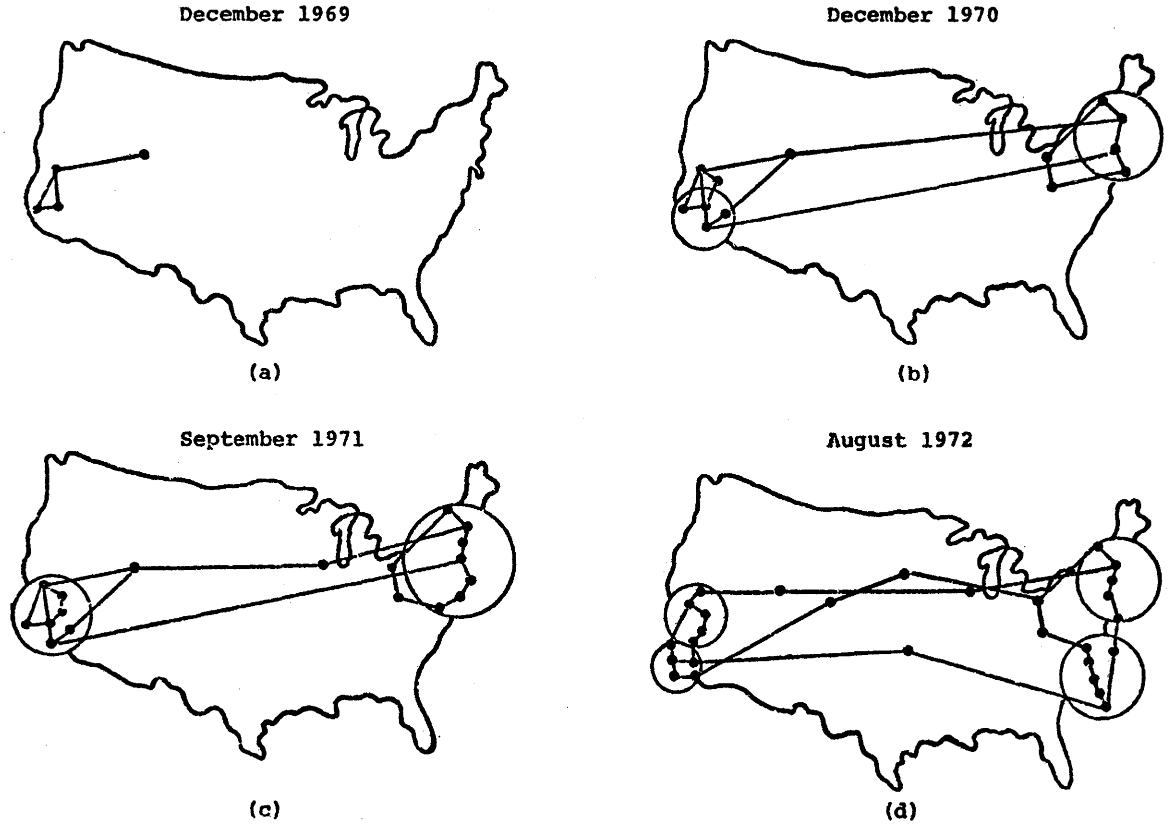


Figure 9—Geographical expansion of the ARPA network

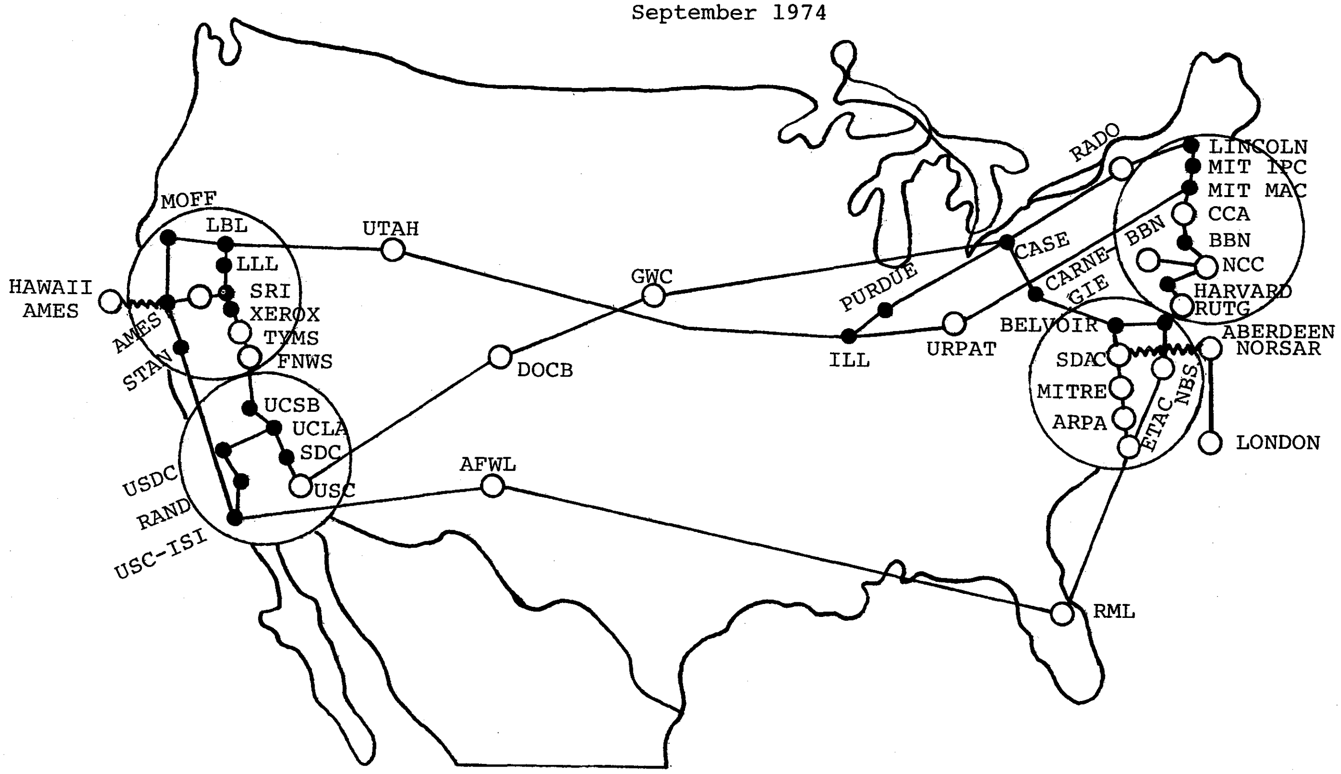
munications functions and other chores needed. Naturally, the communication lines became longer, necessitating communication hardware at the ends of the line.

A result of this approach is star-like networks with a store and forward central switch. The most significant network in this category is the AUTODIN System.[15] AUTODIN was built and is maintained and managed by Western Union for the U.S. Government.

An extension of this type work is the ring computer network in which a front end type device (often called a network interface processor) connects the network lines and the computers. Data for a computer is addressed to that computer and sequentially sent, link by link, in a circular fashion. At each step around the circuit, the data is interrogated by the interface processor and when it finally reaches the interface processor connected to the destination computer, it is removed from the ring. Naturally, if a network like this is not planned very well, data may eventually circulate forever. Thus, control devices to remove data which is "too old" from the network must be placed in the network. In addition, as such a network grows, its reliability can become very low because all elements along the ring must operate for the network to operate. Therefore, additional lines for redundancy and more flexible routing techniques must be added for effective operation.

A more ambitious type of system, called ARPANET,[16,17] was also developing during the last five years. The concept of this system was to provide high flexibility by allowing any kind of interconnections, and adaptive routing of information. In late 1969, the first four elements were installed on the West Coast. The network grew to about a 25 node system in 1971, to about a 40 node system in 1973, and is today about a 50 node system. This network is one of the first major applications of the new technology called "packet switching" in which data is broken up into blocks that are separately addressed and then allowed to make their way independently through the network from origin to destination. This type of network must handle the problem of controlling flows using a "distributed" control scheme.

The ARPANET significantly differs from the centralized system approach. In a centralized system such as NASDAQ, nearly all the controls reside in the central computer. If it cannot handle the flow, the computer will slow down the concentrators and do whatever else is necessary to prevent additional calls from being sent. In a distributed network, very sophisticated techniques of flow control and routing adaption in case of a line or node failure had to be developed. Packet switching is now viewed as a major addition to the technology of computer networking, and has already been applied to radio com-



(e)

Figure 10—Geographical expansion of the ARPA network (continued)

munications.[18] A number of other networks are now being built or designed that are based on the packet switching technology of the ARPANET and the future of the field appears quite bright.

PROPHECY

Clearly an important part of the computer communications revolution has been the proposal and development of an incredible array of digital services. This includes new technical offerings and tariff structures by the common carriers dominated by AT&T[19] and Western Union.[15] A further development of crucial interest to the computer industry is the growth of the specialized common carriers including MCI, DATRAN and a large number of regional carriers such as Western Tele-Communications. The picture is further enhanced by the addition of value added networks and satellite communication. These topics have only been alluded to here since they are the subject of another paper in this session.[20]

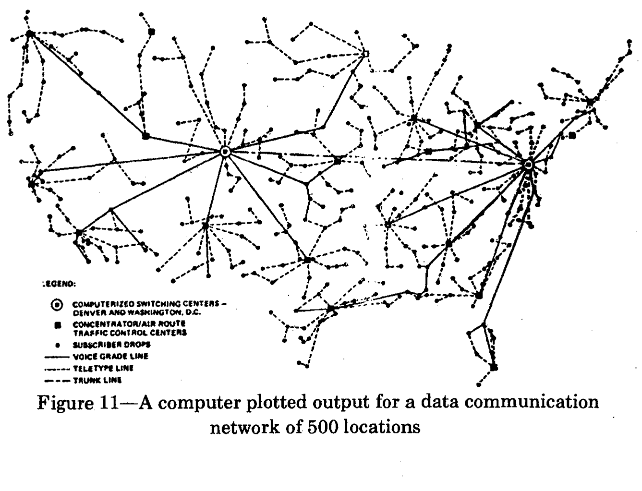

LEGEND:
COMPUTERIZED SWITCHING CENTERS – DENVER AND WASHINGTON, D.C.
CONCENTRATOR/AIR ROUTE TRAFFIC CONTROL CENTERS
SUBSCRIBER DROPS
VOICE GRADE LINE
TELETYPE LINE
TRUNK LINE

Figure 11—A computer plotted output for a data communication network of 500 locations

Our mandate for this talk does not include prophecy—for evil or for good. But after all the only reason for knowing "How we got there", is so we can extrapolate to "Where we are going." Some things are certain. As Fano says "The 'Marriage' of computers and communication has been celebrated and consummated. By now the honeymoon is over and the two partners are beginning to face the realities of their interdependence."[21]

Looking into the very near future networks are planned that tend to combine the distributed network control concepts of ARPANET for computer to computer communications with the centralized NASDAQ-like approach for terminal to computer and terminal to terminal communications. These networks are an extension of the multidrop centralized net where now the terminal processor replaces the computer and the backbone communications is then through a packet oriented net like ARPANET. An example of this type of net is shown in Figure 11. This particular example is a sample design for a planned FAA Air Traffic Control Network. This network has 21 air traffic control computers at appropriate locations. It has a backbone communication network which is a simple loop like network. Emanating from the nodes of this network is an extensive terminal communications network which is itself a collection of networks.

The growth of computer communication networks has clearly left the linear part of its presumed exponential growth. In-house systems or inter-corporation facilities abound not only on paper but in actual implementation. In addition many more facilities are on the horizon. For example:

- In Canada, the Datapac Network is a nationwide, packet switched, shared, data network which has been designed to become the basic Canadian network for data communications. By 1976 there will be four network nodes: Toronto, Montreal, Ottawa and Calgary. These four nodes, or networks switching centers, will initially serve the entire country. By 1980, at least fourteen Canadian cities will have network nodes. After 1980, the network will continue to expand to meet Canada's data requirements.
- Also in Canada plans are being developed for CANUNET, Canadian Universites Computer Network, a packet switched network sponsored by the Ministry of Communications to link some 20 universities.[22]
- An international effort is planned by the Organization for Economic Cooperation and Development. The result is to be a European data communication network between certain universities and research centers. This network, which will work on the "packet switching" principle, is reminiscent of the ARPA network. Secondary networks can be connected to nodal centers. Nodal centers will exist in Italy, France, Switzerland, the United Kingdom, and within the OECD administration. Norway, Sweden, Portugal and Yugoslavia have also joined the project.[23]

Beyond extrapolation we indeed enter the realm of prophecy. We can only list a few achievements we all know are here or on immediate horizon, make an obvious observation, and relate a personal experience.

First the list. The following developments are here:

- Minicomputers
- Programmable calculators
- Hand calculators
- Microprocessors
- Hand held radio transmitters
- Cable TV system for data transmission

Second, the obvious observation. Even without looking into the far future of hand held minicomputers on a chip or optical fibers it is clear that computer networks will soon look nothing like they look now. Mobile users with hand held terminals dialing into vast networks of minicomputers and maxicomputers, with little difference between front ends and processors, is clearly possible.

Finally, a personal experience; as usual, one of us (I.F.) did his Christmas shopping on Christmas eve. He was at the counter at Macy's trying all the calculators, using one calculator to calculate the cost per feature on all the other calculators at the latest bargain price of overstocked Japanese calculators with Italian names. A woman standing next to him, silent for many minutes, finally got up the courage to ask the salesman what memory was used for on a calculator. He tried to explain several times and failed. Finally, he showed her how it was used to store an intermediate answer. A glow of discovery appeared on her face. For the first time after years of propaganda, advertising and intimidation about computer memory banks she understood what memory was. A new American became intimate with the computer. This element of citizen acceptance of the computer when combined with the technical elements make a new revolution both inevitable and unpredictable.

Many others are, of course, actively speculating on the effect of the computer communications revolution on society. Some of this speculation is didactic. Says Peter Goldmark,[24] "What I propose is that the advances of telecommunications technology—satellites, cable TV, broadband circuits and similar devices—make it possible to attract future generations into the smaller towns of America beyond the commuting dependency range of the big city and suburbs and thus cut down on the excessive use of power." Some of the speculation is more ruminative. Says Paul Baran,[25] "The key man in the new power elite will be the one who can best program a computer, that is, the person who makes the best use of the available information and the computer's skills in formulating a problem. In a world where knowledge is power, and where communications mean access to power, he who can most effectively utilize this access will be in the driver's seat. Some persons (primarily computer programmers) claim that the richest man in the world in the year 2000 will be a computer programmer. This may sound outlandish, but few really good programmers laugh when they consider this assertion."

But the best appraisal is by Steward Brand,[26] humanist author of "The Whole Earth Catalog". In his essay, "Fanatic Life and Symbolic Death Among the Computer Bums," he sums it all up, "Ready or not, computers are coming to the people".

REFERENCES

1. Asimov, I., "The Ugly Little Boy," *Nine Tomorrows*, Doubleday & Co., Inc., 1959.
2. Toynbee, A. J., *Change and Habit: The Challenge of Our Time*, Oxford University Press, 1966.

3. Thorndike, L., *A History of Magic and Experimental Science: Volumes III and IV, Fourteenth and Fifteenth Century,* Columbia University Press, Second Printing, 1953.
4. Boorstin, Daniel J., *The Americans: The Democratic Experience,"* Random House, 1973.
5. McLuhan, M. and Q. Fiore, *War and Peace in the Global Village,* McGraw-Hill, 1968.
6. Frank, H., I. T. Frisch and R. Van Slyke, "Testing the NASDAQ System—Traffic and Response Time," *Proceedings of the Symposium on Computer-Communications Networks and Teletraffic,* Polytechnic Institute of Brooklyn, 1972, pp. 577-586.
7. Sackman, H., *Computer System Science and Evolving Society,* John Wiley, 1966.
8. Enticknap, R. G. and E. F. Schuster, "SAGE Data System Considerations", *Transactions of the American Institute of Electrical Engineers,* January, 1959, pp. 824-832.
9. Taplin, J., *A History of the Remote Access Computer Industry in the United States 1964-1972,* Master's Thesis, University of Pennsylvania, August 11, 1972.
10. Knight, J. R., "A Case Study: Airline Reservations Systems," *Proceedings of the IEEE,* Vol. 60, No. 11, November, 1972, pp. 1423-1430.
11. Hargraves, Jr., Robert F., "The Dartmouth Time Sharing Network," *Computer Communication Networks,* N. Abramson and F. F. Kuo (Editors), Prentice Hall, 1974.
12. Gaines, G. and J. Taplin, *Time Sharing Today,* Vol. 3, Nos. 1&2, May 1972.
13. Mauceri, L. J., "Control of an Expanding Network—An Operational Nightmare," *Networks,* Vol. 4, No. 4, 1974, pp. 287-297.
14. Beere, M. P. and N. C. Sullivan, "TYMNET—A Serendipitous Evolution," *IEEE Transactions on Communications,* Vol. COM-20, No. 3, June 1972, pp. 511-515.
15. Cox, J. E., "Western Union Digital Services," *Proceedings of the IEEE,* Vol. 60, No. 11, November, 1972, pp. 1350-1356.
16. Frank, H., I. T. Frisch and W. Chou, "Topological Considerations in the Design of the ARPA Computer Network," *SJCC,* May 1970, pp. 581-587.
17. Roberts, L. G. and B. Wessler, "Computer Network Development to Achieve Resource Sharing," *SJCC,* 1970, pp. 543-549.
18. Abramson, N., "The ALOHA System," *Computer Communication Networks,* N. Abramson and F. F. Kuo (Editors) Prentice Hall, 1974.
19. James, R. J., and P. E. Muench, "AT&T Facilities and Services," *Proceedings of the IEEE,* Vol. 60, No. 11, November, 1972, pp. 1342-1349.
20. Gerla, M., "Moving Bits by Air, Land and Sea," in this session.
21. Fano, R. M., "On the Social Role of Computer Communications," *Proceedings of the IEEE,* Vol. 60, No. 11, November 1972, pp. 1249-1253.
22. deMercado, J., R. Guindon, J. DaSilva, M. Madoch, "The Canadian Universities Computer Network: Topological Considerations," *Proceedings of the First International Conference on Computer Communication,* S. Winkler (Editor), October, 1972, pp. 220-225.
23. Larsson, T., "Data Communication in Sweden—and Some Aspects of the Situation in Europe," *Proceedings of the First International Conference on Computer Communications,* S. Winkler (Editor) October, 1972, pp. 17-25.
24. Goldmark, P. C., "A Rural Approach to Saving Energy," *The New York Times,* Sunday, November 11, 1973.
25. Baran, P., "On the Impact of the New Communications Media Upon Social Values," *Law and Contemporary Problems,* Vol. 34, No. 2, Spring, 1969, pp. 244-253.
26. Brand, S., *II Cybernetic Frontiers,* Random House 1974.
27. Gaines, E. C., "Specialized Common Carriers—Competition and Alternative," Telecommunications, September, 1973, pp. 17-26.
28. Luther, W. J., "Conceptual Bases of CYBERNET," *Computer Networks,* R. Rustin (Editor), Prentice Hall, 1972.
29. Schwartz, M., R. R. Boorstyn and R. L. Pickholtz, "Terminal-Oriented Computer Networks," *Proceedings of the IEEE,* Vol. 60, No. 11, November, 1972, pp. 1408-1422.
30. Worley, A. R., "The Datran System," *Proceedings of the IEEE,* Vol. 60, No. 11, November, 1972, pp. 1357-1368.

Computer communication networks—The parts make up the whole

by WUSHOW CHOU
Network Analysis Corporation
Glen Cove, New York

INTRODUCTION

A computer network, in the broad sense, is any system composed of one or more computers and terminals, communication transmission facilities, and specialized or general purpose hardware to facilitate the flow of data between terminals and/or processors. Its parts consist of communication devices, the host processors, the transmission lines and a set of rules, implemented in either hardware or software, to insure the orderly flow of traffic in the network.

The characteristics of the components of a computer network depend on the environment in which theory is implemented. Thus, in this paper, we first discuss computer network architecture and currently available alternatives of communication devices, transmission facilities, and the required rules and protocols which make up the network.

NETWORK ARCHITECTURE AND NETWORK STRATEGIES

Introduction

In this section, computer communication networks will be classified according to their topological structures and network architecture. A network architecture has at least two levels: the global level of overall networking strategy and the local level of terminal access. In a simple network, terminal access lines constitute the whole network. Thus, there is no difference in the two levels. In more complicated networks, there are communication processors and devices in addition to host computers and terminals. Some or all of the terminals communicate with host computers by first accessing the communication processors (CPs). It is even possible that there are no terminals at all, thus the main characteristics of the overall network may not be the same as that of the terminal access structure.

In the following, the overall networking strategies will be classified as centralized, ring-switched, and store-and-forward (S/F) message or packet switched. The terminal access will be classified as star structured, multidropped, and ring-structured.

Global network architecture

A centralized data communication network

In a simple case, a centralized network may just consist of a computer with a small number of terminals connecting directly to it to form a star structure. Figures 1 and 2 show two more complicated networks structures. A well-known example for the centralized structure is the NASDAQ, which is an over the counter stock automatic quotation system.[1] (Its topological is shown in Figure 1.)

Figure 3 represents a general communication path between a terminal and a host computer. The path has the following sequence: terminal to terminal control unit, to multiplexer, to concentrator, to a front-end processor, to a CP, or a central computer. (Multiplexers and concentrators will be discussed later. For the moment, they may be viewed simply as communications cost saving devices that allow several low speed lines sharing one higher speed line. Multiplexers are usually hardware devices and concentrators are minicomputers.) Not every network or communication path contains terminal control units, multiplexer and/or concentrators.

Centralized network usually has the following typical characteristics:

1. Its computing facilities (i.e., computers) and switching facilities (i.e, if it is a message switching system) are centrally located at one site. However, this statement needs qualifications. In performing the function to allow several low speed lines sharing one high speed line, concentrators do carry out a simple switching function by passing messages between the central computer and the terminals. This switching function which is a necessary consequence of utilizing concentrators is not considered as a real switching function in the above statement. Furthermore, minicomputers used as concentrators and terminal control units are quite underutilized. In some networks, they have been used for limited local processing, local data base access, and/or local switching. These approaches have been termed by some people as "distributed processing" or "distributed data base", even though the basic network architecture is still centralized.

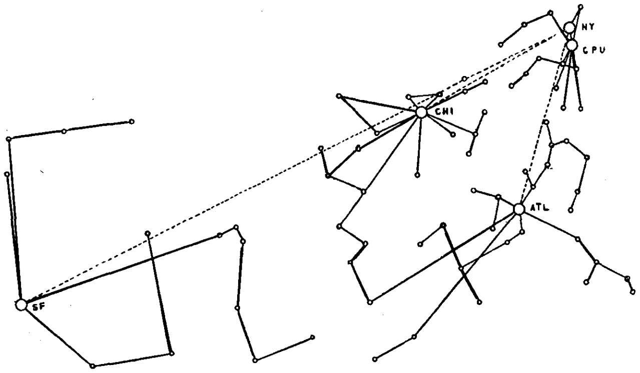


Figure 1—An example of a centralized communication network

2. It has a tree-like appearance. This is quite evident from Figures 2 and 3. However, there are cases in which terminals controlled by the same control unit form a ring or loop-type network. A notable example is IBM's 3600 system.[2]
3. There is only one unique communication path between a terminal and its central computer. (However, there are dial-up lines for backup when dedicated lines fail. Also, there may be parallel lines between two points, such as the central computer and a concentrator for the purpose of higher line throughput and better reliability. In this case, a message may be sorted through any of the parallel lines.)
4. It is a terminal oriented system. Traffic flow is between a large number of terminals and their host computers. There is little or no traffic between computers.

Ring-switched computer network[3,4,5,6]

In a ring-switched network a ring or loop-type network is formed by a set of CPs. Terminals and computers desiring communications are connected to the CPs in a ring or loop. An example is shown in Figure 4. The main function of the CPs (Box B in Figure 4) is to interface the terminals and computers with the ring. They will be appropriately called Ring Interface Processor (RIP) in this paper. More than one terminal and/or computer may be connected to a usually co-located RIP. A RIP bridges its input and output lines with a shift register. RIP switches information from

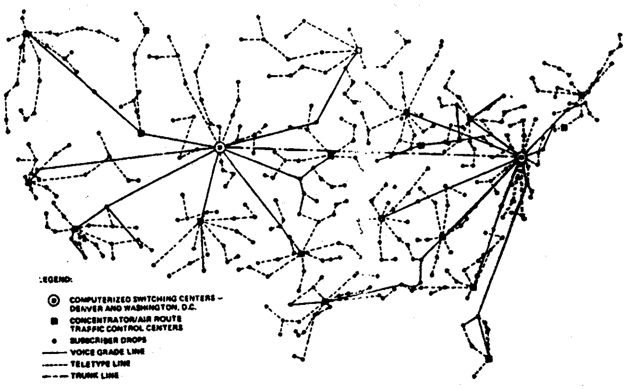

LEGEND:
COMPUTERIZED SWITCHING CENTERS – DENVER AND WASHINGTON, D.C.
CONCENTRATOR/AIR ROUTE TRAFFIC CONTROL CENTERS
SUBSCRIBER DROPS
VOICE GRADE LINE
TELETYPE LINE
TRUNK LINE

Figure 2—Another example of a centralized network

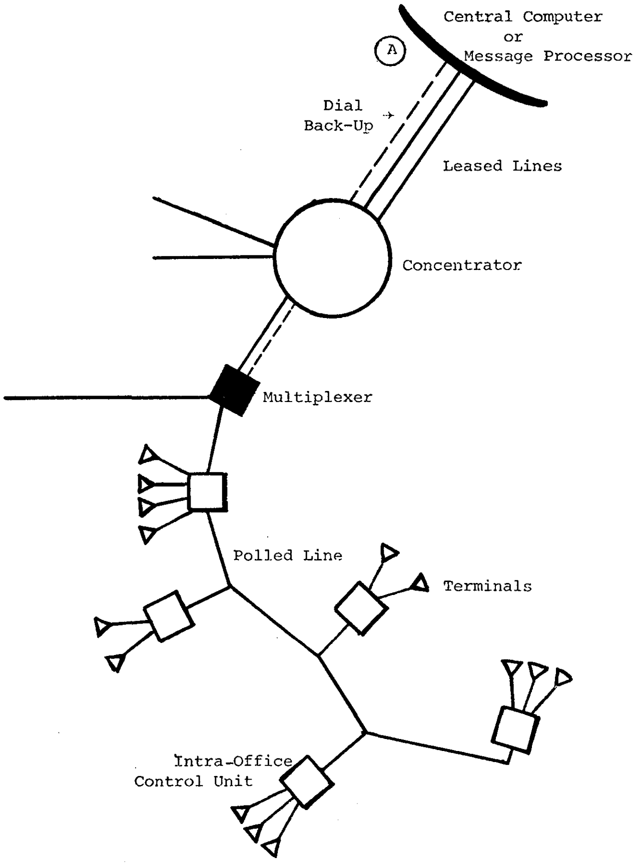


Figure 3—A "general" communication path between a terminal and a computer

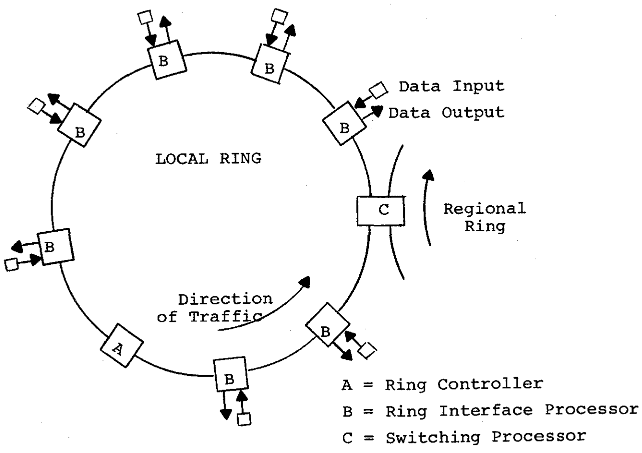


Figure 4—A ring network

input to output by shifting incoming signals from transmission lines through its shift register. The channel capacity of the ring is multiplexed into a series of time slots. (To illustrate, assume the channel capacity is 10 Kbps and it is divided into 10 slots. Each slot will then consist of 1000 bits. Bits belonging to the same slot do not have to be continuous. Without loss of generality, for easy understanding, we can assume they are.) The time slots flow though the ring from RIP to RIP, or from station to station, in the same direction (either clockwise or counterclockwise). When a terminal or computer has a message to send, the message is first stored in the RIP. It is then subdivided into blocks or packets that fit into slots. A header is attached to each packet to indicate the origination and destination. The RIP then checks the shift register and waits for an empty slot. When an empty slot is detected, and available for the RIP to use, the packet is shifted onto the ring to occupy the slot. The RIP also has the responsibility to detect the occupied slots that are addressed to it. Sometimes a minicomputer is included in the ring to perform supervisory functions. A ring-switched network may consist of several rings. Two neighboring rings are interconnected by a switching processor. It transfers from one ring to another by comparing a part of the address included in the packets' header with a wired-in address.

Following are typical characteristics of ring-switched networks:

1. Inexpensive communication hardware
2. Easy to design
3. Low start-up cost
4. High line throughput
5. Low network reliability
6. Higher line costs
7. More suitable for interconnecting terminals and computers in the same building complex than for a transcontinental network
8. Except for terminal access network almost all such networks are experimental
9. Quite often, T1 technology is used for the transmission lines.

Store-and-forward networks

In a message switched, store-and-forward communication network, several geographically distributed processors are linked together with dedicated lines to form a backbone network which is also called the communication subnet. This backbone network acts as a common user service to terminals and computers. Terminals and computers requiring communications must first obtain access at a store-and-forward communication processor. Messages are then sent through the network by the CPs, which in this capacity, are also called switches.

Classical message switching network

Messages are sent in their entirety along a predetermined path from sender to receiver. At each intermediate CP or switch along the path, the message is first stored on an on-line mass storage device or on an off-line storage device (when too long to be feasibly stored within core), and then forwarded to the next CP on the path when an appropriate circuit is available. Compared with packet-switches approach to be described below, the conventional message switching approach has the following disadvantages: very expensive switch costs, long message delays, less efficiency in utilizing network resources and less flexibility in adjusting to traffic conditions.

Packet switching S/F computer network[7]

The basic conceptual difference between message-switching and packet-switching is that in a packet-switching network, a message is subdivided into frames or packets before it is transmitted and is reassembled when it is received. The basic advantage is that the packets can be stored in the main core, instead of in mass storage devices, thus reducing substantially both the delay time and the switch cost.

While the packet-switching concept was being developed, many advanced network control concepts were also developed and new technology utilized. Some of them can be conceptually applied to message switching also. However, people have exclusively associated these new concepts with the packet-switching.

Many of the packet-switching network's desirable characteristics result from the use of adaptive routing, where the path through the network between any two points is not chosen in advance, but is a dynamic function of conditions in the network at any time. With its ability to reallocate its resources as needed, the network overcomes adverse effects of temporary congestion and failed links or switches.

Each switch in the network functions as a "local" network manager, deriving its management information from the network. To send a message, the computer precedes the text of its message with an address and delivers it to its local CP; this minicomputer dynamically determines the best route, provides error control, and notifies the sender of its receipt.

When a message is ready for transmission, the originating CP divides the message into a set of one or more packets, each with appropriate header information. Each packet makes its way independently through the network to the destination CP, where the packets are reassembled into the original message and then transferred to the destination.

A packet-switching S/F computer network provides economical, fast response, and reliable services to its users. However, it is advantageous over other approaches only if there is a large volume of traffic among widespread users.

Terminal access network structures

Star-structure

A Star structure consists of a set of point-to-point connections. Every local access line connects only one terminal to a terminal control unit port, a multiplexer port,a concentrator port, or a computer port. This local access line may be either of the following three connection types:

- Dedicated connection.—Leased line, private transmission line, or hard-wired connection.
- Dial-up connection.—Terminals dial to a multiplexer, concentrator, or computer only if and when there is a need.
- Radio connection.[8]

For the dedicated line connections, each terminal has a fully dedicated port. In the cases of dial-up and radio connection, a set of terminals must share and contend for a smaller number of ports.

Multidrop (or multipoint) line

In this structure several terminals may share one dedicated line (usually leased). A multidrop line has a tree-like appearance. The structure in Figure 2 is composed mainly of multidrop lines. Terminals access the port either by contention or under the control of a computer.

Ring structure

A terminal control unit and terminals are connected in a ring in the same fashion as described earlier. IBM's 3600 system is such an example.[2]

COMMUNICATIONS DEVICES

Communication devices used in centralized network[9]

Multiplexer[10,11]

We will use "facility" to refer to the part of the telephone plant described in terms of its properties as a transmission medium, and "channel" to refer to a functional communications path. A channel is described by its capacity, i.e., the maximum rate at which information can be acceptably transferred over it. The capacity of the channel, or maximum data rate acceptable, depends on a variety of factors, including the bandwidth of the facility and the hardware characteristics of the modems. The use of one facility to form several separate channels is called multiplexing. A device which combines multiple facilities, each used for one or more distinct channels, into one facility, formed into the same distinct channels, is called a multiplexer. A device performing the reverse process, i.e., transforming one facility, formed into several channels, into multiple facilities, each with one or more of the channels, is called a demultiplexer. Many current hardware devices perform multiplexing in one direction, and demultiplexing in the other direction. Such a device is usually simply called a multiplexer.

The channel is the functional communications path, whereas the facility is part of the hardware used to form a channel. A multiplexer does not alter the channel structure of the network, and thus is functionally transparent. However, the physical facilities from which channels are formed determine a large part of network costs. Multiplexing offers a way to achieve significant economies in facilities use. To understand these economies, it is helpful to examine the two fundamental approaches to implementing multiplexing.

One approach is to divide the bandwidth of the facility into several separate segments, and allow each segment to serve a separate channel. This is referred to as frequency division multiplexing (FDM). The second approach is to establish a high speed data stream over the facility and assign periodic time slots or bits positions of the data stream to separate channels. This is referred to as time division multiplexing (TDM). There are several variations on the implementation of these approaches.

Concentrator[10,12,13]

The word "concentration" appears to have a very broad meaning in data communications. We will discuss only one narrow interpretation of concentration.

Consider a device having several facilities connected to its input, and only one facility connected to its output. At this point the device may be a multiplexer. However, it is distinguished by the following characteristic: the single facility on the output side carries one channel, the capacity of which is less than the sum of all the capacities on its input side. Such a device providing effective communicatons is called a concentrator. A multiplexer is transparent to the channel structure of a network; a concentrator obviously is not.

The percent of time a channel is used is called its utilization. Many terminals generate data for transmission at an average rate which is much less than the capacity of the channel; resulting in channels with low utilization. A concentrator achieves economic advantage by replacing several low utilization channels with one highly utilized channel. A prerequisite for a concentrator is that its output channel capacity be greater than the sum of the average data rates of the terminals on its input. It is at this point perhaps helpful to examine the difference between a multiplexer and a concentrator in more detail.

To each time slot of each channel on the input of a TDM, a time slot is assigned in the high capacity channel on its output. This effectively divides the high capacity output channel into several separate subchannels, each associated with a particular channel on the input. It does not matter whether or not a time slot is being used to transfer information. A concentrator has more time slots arriving on its input side than leaving on its output side. Each time slot carrying information must be assigned a time slot on the output side. Thus a concentrator must be able to identify which time slots are in fact transferring information. Furthermore, it must be able to assign output time slots to this information in such a manner as to be understood by whatever device is on the other end of the output channel. Although the average number of time slots carrying information on the input will be less than the number available on the output, the random nature of terminal use may result in the number of slots carrying information arriving over a brief interval being greater than the number of slots available on the output. Hence, the concentrator must also have the ability to buffer the arriving information as it waits for available slots. The requirements of intelligence and storage for a concentrator invariably lead to its implementation with a minicomputer. The actual operation of concentrators varies considerably, but is usually much more sophisticated than the simple bit packing noted above. By performing such local operations as polling, error checking, line control, etc., and transferring information to the computer with efficient high speed transmission techniques, the concentrator can achieve an apparent output channel utilization in excess of 100 percent.

The minicomputer implementation of a concentrator implies a fundamental component cost of approximately $10,000. Compared to a $1000 cost of a multiplexer, such a figure requires large economies to be achieved for cost effectiveness. Concentrators can typically handle 64 channels, (provided reasonable traffic characteristics). However, hardware required in addition to the minicomputer to achieve this capability raises the cost to approximately $20,000, or $500/month rental (excluding maintenance). (Dollar values are for the purpose of illustration only.)

Front end processors

The central computer and terminals use the data communications network to interchange information. The general facility of a computer for transferring information between it and the outside world is its input/output (I/O) channel. Particular devices are connected with a hardware interface. In the case of a communications line, the modem terminating the line must be interfaced with the CPU. The overhead required for a large CPU to interact with many communications lines at a modem level is far too great to be economically attractive. Thus a sophisticated interface is used to handle the modem interaction, and only useful information is transferred through the I/O channel to the CPU. In the early history of such interfaces, hardwired logic devices called Line Termination Units (LTU) were used. More recently, it has become very attractive to use minicomputers to accomplish this task. Such minicomputers are called Front End Processors (FEP).

The software capabilities of minicomputers results in a very broad range of sophistication in their use as FEPs.

Other devices

Modem[14]

A voice-grade line may be roughly characterized as having usable bandwidth extending from 300 Hz to 3400 Hz. Full duplex lines can transfer information simultaneously in both directions, while half duplex lines can transfer information in only one direction at a time. Both computers and terminals supply and accept information in the form of a digital baseband signal. The function of the modem (modulator-demodulator), is to interface the digital baseband requirement to the analogue bandpass requirement.

Modem sharing unit

A modem sharing unit (MUS), or multiple access coupler, is a device for connecting several (typically up to six) terminals to a single modem. The terminals are usually restricted to be in the same location (within 50 feet of the MSU).

Port sharing unit

A port sharing unit (PSU) is a device for connecting several (typically up to six) modems to a single computer port (or conecentrator, or multiplexer). The PSU broadcasts data from the port to all the modems, and delivers data to the port from the first modem to generate an appropriate response.

Biplexers

A biplexer is a device which uses two voice grade lines to effectively achieve a single high speed channel (up to 19.2 kbps). Such a device must be able to compensate for the possible differential delays of the two separate facilities. Typically, acceptable operation can be achieved with the two lines diversely routed with differential delays up to ½ second.

The cost effectiveness of a biplexer is principally derived from the current tariff structure for high speed lines versus voice grade lines.

Communication devices for packet switching S/F networks

There are three major communication functions that CPs in a S/F packet-switching network must perform: interfacing host computers with the backbone network, interfacing terminals with the backbone network, and managing the packets flowing through the backbone network. It is not necessary to have three distinctive types of CPs to handle these three functions. It is possible to have one type of CP perform more than one of the functions. On the other hand, a host computer or a remote concentrator may perform part of the interfacing functions also.

Interfacing host computers with the communication subnet.

This function includes the following tasks:

1. Breaking a long outgoing message into message blocks so that the size or length of the message

blocks are within the limit allowed by the network's protocols. (In the ARPANET, it is about 8000 bits and is simply called as a message rather message block)
2. Formatting and code-converting the message blocks into a standard format acceptable to the network
3. Attaching a header with addresing and control information to each message block. (In ARPANET, this header is called a "leader")
4. Attaching a trailer with error checking information to each message block
5. Storing the unacknowledged messages and/or message blocks for possible retransmission
6. Reassembling receiving message blocks into messages
7. Breaking long outgoing message blocks into packets or frames so that the maximum length of the packets is within the limit acceptable to the network. (In APANET, it is 1008 bits)
8. Attaching a header to each packet
9. Attaching a trailer to each packet
10. Storing the unacknowledged packets for possible retransmission
11. Reassembling receiving packets into message blocks.
12. Controlling the input rate to avoid congestion.

This interfacing function can be carried out in two possible ways:

1. Tasks 1 through 6 are carried out in host computers by attaching to host computers with appropriate hardware interface and software interface. Tasks 7 through 12 are carried out as part of communication subnet switches' responsibilities. (This approach has been adopted by ARPANET. Host computers are linked directly to Interface Message Processors, IMPs, the name for the switching nodes in the ARPANET, with host computers performing the first 6 tasks and IMPs performing the last six tasks.[15,27])
2. All the tasks are lumped into one CP. (If so, some of the tasks can be merged, to reduce the total number of distinct tasks.) In this case, there will be an additional processor between a switching node and its host computer and when the switching node receives a packet, it will not know whether the packet is from a host computer or a neighboring switching node.

Interfacing terminals to the communication subnet.

The following are some of the tasks belonging to this function:

1. Recognizing functional applications of the messages, such as file transfer, interactive, RJE, etc., such that appropriate protocols can be applied.
2. Breaking long messages (RJE, graphic terminals, etc.) into packets.
3. Code-conversion for a variety of different terminals.
4. Formatting the packets.
5. Attaching the headers and trailers.
6. Storing unacknowledged message or packets for possible retransmission.
7. Reassembling receiving packets.

This function can also be achieved with two approaches:

1. Group this function with the traffic managing function, (i.e., make it part of the switching node's responsibilities). (In ARPANET, the CP performing both functions is called a Terminal IMP, or TIP.[16])
2. Make a distinct CP, specially designed to handle all tasks. This CP will stand between switching nodes and terminals. In this fashion, when a switching node receives a packet, it may not know whether the packet is from a terminal or a neighboring switching node. The CP in this case may act like a front-end to the switching node, a host computer, or a concentrator. (In ARPANET, there is a minihost to interface RJE terminals and IMPs, there is a special communication processor called ELF to interface a variety of terminals with the IMPs. ELF is a PDP-11 based system.)

Managing packets in the network

This is the most important function in an S/F packet switching network. Among the tasks that can be classified into this function are:

1. Routing input packets to appropriate output lines according to packet destination, traffic condition and routing tables.
2. Periodically updating routing-tables.
3. Detecting network element failures and network disconnection.
4. Controlling input rates to avoid traffic congestion.
5. Recovering from failure.
6. Acknowledging packet receipt.
7. Controlling errors
8. Statistics collection

There are four possible ways to perform this function:

1. Combining this function with part of the host computer interfacing function (e.g., ARPANET's IMP).
2. Combining this function with the terminal interfacing function (e.g., ARPANET's TIP).
3. Combining this function, the terminal interface function, and part of the host interface function into one machine (e.g., the High Speed IMP (HSIMP) also called the pluribus IMP, currently being developed for ARPA by Bolt Beranek and Newman, Inc.[17])

4. Performing no other functions but managing the packet flow.

Communication device for ring-switched network[5,18,19]

There are three major communication functions in a ring-switched network: ring interfacing, ring control, and switching between a ring and the rest of the network. Depending on design philosophy, the control function may be distributed among RIPs and switches.

Ring-interface

A RIP basically consists of a shift register, buffers and an associative store which can be written into by the attached computers or terminals. A RIP can be a minicomputer, microprocessor, or hard-wired device. Among the tasks to be performed are:

- Breaking messages into packets
- Detecting a usable empty slot for sending packets. (An empty slot may not be usable. In a central control system, the assignment of empty slots to users is the responsibility of a central controller)
- Shifting packets onto the ring
- Detecting arriving packets
- Shifting arrived packets into buffers.
- Error control
- Erasing delivered packets from ring slots, if this function is not performed by the ring-controller.

Ring-control

Major tasks performed by a CP designated as the ring controller are:

- Maintaining synchronization of the ring
- Preventing the build up of traffic in the ring because of undeliverable packets. (If a packet tries to pass through the controller a second time, it is either destroyed, creating an empty slot or sent back to its destination)
- Empty slot assignment upon demand. (This is performed only in a centrally-controlled system in which a RIP cannot shift a packet into an empty slot without permission from the controller.)

Switching function

Packets destined for a station outside a particular ring have addresses indicating this and are picked off by a switching node in exactly the same way that intra-ring traffic is picked off by the RIPs. This traffic is buffered and shifted onto the next ring in the same way that local traffic is shifted onto a ring by the RIPs.

TRANSMISSION FACILITIES

Transmission signaling

There are two ways in which digital signals are sent down a transmission line. They may be sent as they are, without modulation, or they may be superimposed upon, or "modulate" a higher frequency which "carriers" them. Without modification, signals cannot be sent long distance because of distortion on the line. If the data signals are to be carried by a high frequency, they may either be sent in analog or digital form.

1. Basehand transmission
   The transmission of signals at their original frequency and shape is called "basehand" signaling. Basehand signals may be sent over open wire pairs of a few miles in length at speeds up to 300 bps. The speed could be increased significantly if coaxial cables rather than open wires are used, or if regenerative repeaters are inserted at approximately one thousand foot intervals in the line.
2. Analog transmission
   Almost every data and computer communication network relies on the telephone plant's facilities for transmission. Today's telephone facilities have been designed for voice transmission and almost all of them use analog transmission with frequency division multiplexing, requiring analog modulation for carrying and transmitting data signals. A voice grade channel has a bandwidth of 4K Hertz (cycles per second). With proper modulation techniques, up to 4.8 Kbps can be derived from a dial-up line and with proper line conditioning, up to 9.6 Kbps can currently be derived from a voice grade leased line. It is possible that even higher rates will be achieved in the future. Data rates commonly derived from a voice channel are 300 bps, 600 bps, 1200 bps, 1800 bps, 2000 bps, 2400 bps, 3600 bps, 4800 bps, 7200 bps and 9600 bps. For speeds higher than this, broadband channels are necessary. At present, modulation of 6 voice band channel yields 19.2 Kbps, modulation of 12 voice band channels yields 40.8 or 50 Kbps, and modulation 60 voice band channel yields 23.4 Kbps.
3. Digital Transmission (or pulse transmission)
   With digital transmission (in contrast to analog), a train of high rate pulses is used to "carry" information (digital information as well as voice), instead of a sinusoidal, or analog carrier. A commonly used technique is called Pulse Code Modulation (PCM). In the Bell System, this carrier system is called T1 carrier. It has a total rate of 1.544 Mbps and is multiplexed to several lower rate channels. Using the T1 system, a voice grade channel is equivalent to 56 Kbps (in contrast to 9.6 Kbps in the analog system), and, is thus much more economical for digital information transmission. Transmission offerings are discussed in detail by Gerla in another paper in these proceedings.[20]

Terminal access lines and trunk lines

Terminal access lines

With the exception of RJE and graphics terminals, terminal speeds are rarely higher than 2400 bps. Speeds of RJE and graphics terminals, are usually no greater than 4800 bps. Thus a terminal access connection is usually a subvoice line, voice grade line with medium speed modem, dial-up connection, or 2.4/4.8 Kbps DDS line. Exceptions are for terminals using radio wave links and using the ring switching technology.

Trunk lines

Trunk lines are the lines on the higher speed side of multiplexers and concentrators, the lines connecting CPs, or the lines connecting host computers to the CPs.

1. Centralized network
   The speed of lines merging from a multiplexer or a concentrator is usually 2.4, 3.6, 4.8, 7.2, or 9.6 Kbps. Thus, voice grade or DDS line can be used. Occasionally, higher speed lines, 19.2 Kbps or 50 Kbps are also used between concentrators and host computers.
2. Ring-switched networks
   T1 carriers with their capacity of 1.544 Mbps are often used, even though other types of services and speeds can also be used.
3. S/F networks
   Trunk line speeds range upwards from 9.6 Kbps. For example, in ARPANET, most inter-IMP/TIP lines have a speed of 50 Kbps, with a few 230.4 Kbps lines. The line speed between a host computer and its IMP or TIP is usually 100 Kbps. Thus, most suitable lines to be used for S/F networks backbone trunks are broadhand lines, Telepaks and DDS 56 Kbps lines.

TRAFFIC MANAGEMENT AND SOFTWARE

Introduction

In any communication system, and in particular, a computer communication system, it is essential to have a set of well designed basic control procedures to insure efficient, correct and smooth transfer of information in the system. The main purpose of these control procedures are:

1. To make the system convenient to use
2. To prevent loss of data
3. To detect message duplications
4. For efficient and orderly use of resources (lines, communication processors, etc.)
5. For error detection and correction
6. To detect system element failures
7. For recovery from system failure
8. To prevent and recover from traffic deadlocks
9. To prevent congestion

In a centralized system, these procedures are relatively simple. On the other hand for packet switched S/F systems, they are very complex and must be carefully designed. In general, they can be partitioned into four catogories.

1. Communication Protocols
2. Flow control strategies
3. Routing strategies
4. On-line monitoring and control

A communication protocol is a set of rules established to manage the information exchange between two communication entities. (For example between a pair of communication processors.) The protocol provides standard representations to allow the communicating entities to understand one another and to cooperate with one another. The goal of such rules, is to insure that the information exchange is made in an orderly fashion. A flow control strategy is a set of rules that governs the acceptance of traffic data into the system, or into a communication processor. The design objective for such rules is to optimize the trade-off between traffic congestion protection and system performance during normal traffic conditions. A routing procedure manages output queues in the communication processors. It decides when and where a message should be transmitted. The design objective of such procedures is to minimize the message delay and optimize the throughput according to traffic conditions. Strictly speaking, flow control and routing are a subset of the communication protocols. However, because of the unique functions and importance of flow control and routing, they will be treated separately in this paper. An online monitoring and control function monitors system malfunctions and performance, and generates diagnostic information. This function provides the means to report system changes and malfunctions so that correct measures can be taken when needed.

Protocols

In a terminal oriented centralized system, there are in general two levels of protocols: (1) the line control procedure that administers the physical transmission medium and, possibly, detects and corrects errors; (2) the protocol that manages the information flow between a terminal and a concentrator or a host computer. In a packet switched system there are three additional possible levels of protocol between the two. The protocols are further complicated by the requirements that there are in general a variety of communication functions to be performed among diverse terminals and computer operating systems.

Protocols can be classified into the following five levels.

Not every computer network has all of them:[21,22]

- Line control procedures.
  This is the lowest level of all of the protocols. Such a protocol administers the physical transmission medium and, possibly, automatically detects and corrects errors (such as by retransmission). ANSI's ADCCP, IBM's BSC and SDLC,[23] and ISO's HDLC all belong to this level.
- Between a pair of communication processors (such as IMPs). This protocol provides for reliable communication among communication processors and handles transmission error detection and correction, flow control, and routing.
- Between a communication processor and a host computer, or between a communication processor and a terminal. With this protocol, a host computer (or terminal) has operating rules that permit it to send messages to specified host computers (or terminals) and to be informed of the disposition of those messages. In particular, it constrains host computers (or terminals) to make good use of available communications capacity without denying such availability to other users.
- Between a pair of host computers. This set of rules allows host computers to maintain communications between processes (user jobs) running on remote computers. One process requiring communications with another on some remote computer system makes requests on its local supervisor to act in its behalf in establishing and maintaining these communications under this protocol.
- Between a pair of user processes (e.g., a terminal and a time-sharing operating system). This is the highest level of protocol, the user level. It provides user processes (modules in time sharing computer systems, modules in multiprogramming systems, terminals) with a general set of primitives to isolate them from many of the details of operating systems and communications. At this user level, the protocols are interface function oriented and join an open-ended collection of modules. Examples are remote job entry protocol, file transfer protocol, etc.

Flow control

Flow control procedures regulate the input amount and rate a communication processor can accept in order to prevent or minimize the occurrence of traffic congestion and deadlocks. In a centralized system or a ring-switched system, the flow control procedure is very simple. For the former, the concentrator or host computer stops polling terminals if no appropriate buffers are available for an input. For the latter, no message or only a few messages can get on to the ring, if the ring is fully or highly utilized.

For S/F systems, flow control procedure must be very sophisticated and is still an ongoing topic for research. Many flow control strategies have been proposed. Some have been implemented. Although they are different, they basically achieve the control by allowing an input message only if a buffer has been reserved for it, and/or by limiting input to communications processors if the number of occupied buffers reaches a specified lower limit.[26,24]

Routing

The routing problem is a centralized network or in a ring switched network is elementary. This is usually one unique path from origination to destination. In the centralized network, a message originating from a terminal is routed to a concentrator (if there is one) over the only line between the two, then to the host computer. (Sometimes there are parallel lines between a concentrator and a host computer. The choice is then the first non-busy line for routing.) In the ring network, a message placed on the ring circulates in a specified direction until it reaches its destination if both origin and destination are on the same ring. Otherwise, it circulates on ring to a switching node, where it is switched to a different ring and then circulates on the new ring in a similar fashion.

For the S/F computer network, the routing problem is more complicated. The existing and proposed routing strategies can be characterized as follows.[25]

Deterministic vs adaptive

- In a deterministic routing strategy, packets between a same pair of communication processors are always routed through a same path, unless there is network element failure.
- With an adaptive (or dynamic) routing strategy, messages flowing between a pair of processors are not necessarily routed through a same path. The chosen path usually has certain desirable characteristics, such as least delay or maximum available capacity and varies according to traffic conditions.

Centralized control vs distributed control

- Centralized control: Determination of host routes is performed at one computer.
- Distributed control: Best routes are determined at each communication processor.

Single path vs multiple paths

- Single path: There is only a single path that can carry traffic between a same pair of communication processors.
- Multiple paths: At any time traffic may flow through more than one distinct path between the same pair of communication processors.

On-line monitoring

This function is sometimes termed an on-line control function and is not necessarily less complex in a centralized network then in a S/F computer network. Generally speaking, this function incorporates the following elements:

- Monitoring and reporting malfunctions in the communication network, terminals, controllers, data base, etc.
- Performance monitoring and interpretation for response time and traffic activities.
- Generating diagnostic responses for input transactions that could not get a normal response.

CONCLUSION

Based on a broad definition for the computer network, three basic types of computer networks are defined and characterized. Centralized, ring-switched, and store-and-forward switched. For each type, the parts that make up a computer network are given. The parts are: communication devices, transmission facilities and traffic management. Communication devices carry out the responsibilities of switching, network control, interfacing, and/or saving communications costs. Transmission facilities interconnect communication devices, terminals and computers. Depending on applications, their speed ranges from under 100bps to over 1.5Mbps. Traffic management is a set of rules that ensure the smooth and orderly exchange of information among elements of a computer network. Its main functions are protocol, routing, flow control and monitoring. In short, this paper has explained "what a computer network is and what it consists of."

REFERENCES

1. Schwartz, M., R. R. Boorstyn, and R. L. Pickholtz, "Terminal-Oriented Computer-Communication Networks," *Proceedings of the IEEE,* November 1972, pp. 1408-1423.
2. IBM, "IBM 3600 Finance Communication System—System Summery," *GC27-0001-2,* 1973.
3. Pierce, J. R., "How Far Can Data Loops Go," *IEEE Transactions on Communications,* Vol. COM-20, No. 3, June 1972, pp. 527-530.
4. Farmer, W. P. and E. E. Newhall, "An Experimental Distributed Switching System to Handle High Speed Aperiodic Computer Traffic," *Proc. ACM Symp. Problems on the Optimization of Data Communication Systems,* 1969.
5. Farber, D. J. and K. Larson, "The Structure of a Distributed Communication System," *Proc. of the Symp. on Computer-Communications Networks and Teletraffic,* Polytechnic Institute of Brooklyn, April 4-6, 1972.
6. Fraser, A. G., "Spider—An Experimental Data Communications System," *Proc. of 1974 NTC.*
7. Roberts, L. G. and B. Wessler, "The ARPA Computer Network Development to Achieve Resource Sharing," *Proc. of the SJCC,* AFIPS Press, 1970, pp. 543-549.
8. Abramson, N., "The ALOHA System," *Technical Report B72-1,* University of Hawaii, January 1972.
9. McGregor, P., "Effective Use of Data Communications Networks," *Proc. of the NCC,* AFIPS Press, 1974, pp. 565-575.
10. Doll, D. R., "Multiplexing and Concentration," *Proceedings of IEEE,* 60:11, November 1972, pp. 1313-1321.
11. Pack, C. D., "The Effects of Multiplexing on a Computer-Communications System," *Communications of the ACM,* Vol. 16, No. 3, March 1973, pp. 161-168.
12. Newport, C. B. and J. Ryzlak, "Communication Processors," *Proceedings of IEEE,* Vol. 60, No. 11, November 1972, pp. 1321-1332.
13. Mills, D. L., "Communication Software," *Proceedings of the IEEE,* Vol. 60, No. 11, November 1972, pp. 1333-1341.
14. Davey, J. R. "Modems, " *Proceedings of IEEE,* 60:11, November 1972, pp. 1284-1292.
15. Heart, F. E., *et al.,* "The Interface Message Processor for the ARPA Computer Network," *AFIPS Proceedings, SJCC,* Vol. 36, 1970, pp. 551-567.
16. Ornstein, S. M., *et al.,* "The Terminal IMP for the ARPA Computer Network," *AFIPS Proceedings, SJCC,* Vol. 40, 1972, pp. 243-254.
17. Heart, F. E., *et al.,* "A New Minicomputer/Multiprocessor for the ARPA Network," *Proceedings of the NCC,* 1973, pp. 529-537.
18. Hayes, J. F. and D. N. Sherman, *Bell System Technical Journal,* Vol. 50, No. 9, November 1971, pp. 2947-2978.
19. Hassing, T. E., R. M. Hampton, G. W. Bailey, and R. S. Gardella, "A Loop Network for General Purpose Data Communications in a Heterogeneous World," *Proc. of 3rd Data Communications Symp.,* November, 1973.
20. Gerla, M., "Moving Bits by Air, Land and Sea: Carriers, Vans and Packets," *Proceedings of 1975 NCC.*
21. Crocker, S. D., *et al.,* "Function-Oriented Protocols for the ARPA Computer Network, " *AFIPS Proceedings, SJCC,* 1972, pp. 271-279.
22. Trans-Canada Telephone System, "Datapac Standard Network Access Protocol," 1974.
23. IBM, "IBM Synchronous Data Link Control—General Information," *GA27-3093-0,* 1974
24. Opderbeck, H. and L. Kleinrock, "The Influence of Control Procedures on the Performance of Packet-Switched Networks," *Proceedings of 1974 NTC.*
25. Gerla, M., "Deterministic and Adoptive Routing Policies in Packet-Switched Computer Networks," *Proc. of 3rd Data Communications Symp.: Data Networks Analysis and Design,* November 1973.
26. Gerla, M. and W. Chou, "Flow Control Strategies in Packet Switched Computer Networks," *1974 NCC Proceedings,* Volume 43, AFIPS Press, Montvale, N.J.
27. Carr, S., S. Crocker and V. Cerf, "Host-Host Communication Protocol in the ARPA Network," *AFIPS Conference Proceedings,* Volume 36, 1970, AFIPS Press, Montvale, N.J.

Moving bits by air, land and sea—Carriers, vans and packets

by MARIO GERLA and JOHN ECKL
Network Analysis Corporation
Glen Cove, New York

INTRODUCTION

In the past, the data network designer was confronted with only a few choices relative to communications service alternatives and line tariffs. Basically, he had to choose between dial-up or private line from terminal to computer and, in the case of private line, between narrowband or voice grade. After this preliminary choice was made, the network was optimized based on a well defined line cost structure.

Recently, the need for high bandwidth and high quality computer to computer communications, and the emergence of new communications services both from conventional common carriers and from specialized carriers has created new requirements and new line alternatives for the data network user, thus adding a new, important dimension to network design. Line economy, service quality, network growth flexibility and value added services are among the considerations that should guide the user in the selection between such alternatives.

In this paper, we attempt to identify the impact of the new offerings on the optimal network strategy, in a typical data communications environment. The important aspects of the various alternatives are briefly outlined and compared, and some technical details on the operation of the new value added networks are presented. General guidelines for the selection of the best alternative are provided, and are illustrated in two applications.

NETWORK STRATEGIES

In selecting the optimal strategy for a data network configuration we must consider a variety of elements such as: number and location of terminals and host computers; terminal speed; traffic pattern (single or multiple hosts); traffic volume; type of data transmitted (interactive, file transfer, computer to computer etc.); terminal connect time and frequency of usage; reliability requirements etc.

In some cases the best strategy might be that of connecting the terminals to the host computer via dial-up; in other cases a high speed, distributed network is required to interconnect computers and terminal concentration sites (e.g., ARPANET). It is clear that the new carrier services will have a different impact on the two above mentioned limiting cases. More generally, each carrier service has a different impact on different types of data communications users, network strategies and requirement profiles. Therefore, a rigorous cost-performance comparison of the various offerings would imply the analysis case by case of an extremely large number of possible situations, and is certainly beyond the scope of the present paper. Instead, we focus here on a typical network structure and evaluate the impact of the new offerings on its cost and performance.

The most general configuration of a modern, medium-sized, nationwide, terminal oriented data network is represented by a two-level hierarchical structure.

The lower level corresponds to several *local distribution* subnetworks which connect geographically distributed terminals to regional collection centers (which could be TDMX devices, concentrators, packet switching processors, satellite ground stations etc.). A variety of techniques can be used for connecting terminals to regional centers, such as: dial-up, time or frequency division multiplexing and polling.

The higher level network is the *backbone* network which connects the regional centers to the host (or hosts) and, if required, between each other. We can identify two types of backbone configurations: the tree-like structure (star, minimum spanning tree, or an intermediate solution) generally used when there is only one host; and the distributed, 2-connected packet or message switched structure, which is desirable when there are several hosts in different locations, or when two disjoint paths to the host are required for reliability. Backbone links are generally implemented with synchronous channels of voice grade bandwidth or higher.

The above network model includes the two limiting cases of a nationwide computer to computer network (in which case the network reduces to the high speed backbone component) and of a local, single host, terminal oriented network (in which case the network reduces to the local distribution subnet).

The emphasis of the new offerings is to provide a very competitive (cost-wise) and specialized (quality-wise) service on long distance routes between a selected number of cities, with only a limited capability of extending such service outside the urban areas. Therefore, the user can achieve considerable savings in backbone trunk cost if the regional centers are properly selected and, in the limit, can even consider to replace a private backbone network with the shared network facilities of value added vendors such as PCI and Telenet. However, he should be aware of the fact that line cost saving, quality improvement and flexibility gained in the backbone network must be traded off, in general, with a cost increase and performance degradation in the local distributions, especially if a substantial number of his terminals require interconnection via high cost and low quality conventional communication services.

CONVENTIONAL AND SPECIALIZED CARRIERS

In the following we briefly review the properties of conventional carrier (AT&T, Western Union), specialized carrier (MCI, Southern Pacific Communications, U.S. Transmission Systems, Western Telecommunications, Datran) and satellite carrier (American Satellite, RCA, Western Union) offerings; and relate such properties to the line cost economies obtainable during network design.

Switched services

The use of dial up over the public switched network is cost-effective both in the local distribution subnetwork (primarily as a local access technique for terminals of unfrequent use) and in the backbone network (primarily as a back up in case of failure of the leased facilities).

Dial up is presently offered by the common carriers over half duplex, non-conditioned voice grade facilities. The quality is acceptable for low speed connections, but is often not adequate for backbone communications. In fact, full duplex operations require two half duplex lines, and data speed cannot exceed 4.8 Kb/s.

A new type of truly digital switched service (Data Dial) will be soon offered by Datran, and probably AT&T and other specialized carriers will follow the example. The major features of this service are: digital channel (no modems); connection established in less than 1 second; large selection of channel speeds (up to 19.2 Kb/s) at different costs and with 1 second incremental charges; low error rate; low blocking.

The introduction of the digital dial up might have a substantial impact on distributed network strategies. In particular, it could efficiently complement, if not replace, the packet switching strategy to accommodate bursty, high speed distributed requirements in a network with several computers and high speed peripherals that can communicate with each other.

Dedicated line offerings

The majority of data communications services available today are based on analog facilities originally developed for voice communications. The most popular voice grade dedicated service is offered by AT&T under the Hi-Lo Tariff, and can accommodate up to 9.6 Kb/s with line conditioning and appropriate modems. The Hi-Lo rate structure is a location dependent structure, in the sense that it applies different mileage and service termination charges depending on the locations of the stations at both ends of the line. More precisely, there are 370 locations, classified as high density locations and corresponding to high volume communications areas, while the remaining locations are designated as low density. The basic elements of the tariff are reported below:

High point-high point:	.85 \$/Mile × Mo.
High point-low point or low point-low point:	2.50 \$/Mile × Mo.
Short haul (≤25 miles):	3.00 \$/Mile × Mo.

Monthly channel terminal charges are \$35 for Hi and \$15 for Low; station terminal charges are \$25 for both Hi and Low.

A low to low connection can be implemented either directly (in which case the low to low direct distance charge applies), or via two intermediate high density points (in which case different tariffs apply to different segments). For a typical data network with geographically distributed terminals, the effect of the Hi-Lo structure (as opposed to a uniform structure) is that of reducing backbone cost, at the expense of higher distribution costs, especially if most of the terminals are in areas with low degree of industrialization.

For data rates higher than 9.6 Kb/s, AT&T offers the Series 8000 service for speeds of 19.6 and 48 Kb/s, and the Series 5000 service (Telpak) for speeds up to 230 Kb/s. Both offerings are based on analog channels, which can be subdivided into lower speed, voice grade channels.

In addition to the analog facilities, AT&T plans to offer in the near future the Dataphone Digital Service (DDS), a truly digital service, with synchronous transmission at speeds of 2.4, 4.8, 9.6 and 56 Kb/s. Main features of the system are: end to end digital transmission (no modems required); high circuit availability (99.9 percent); and mileage charges considerably lower than the analog channels of equivalent bandwidth. The basic elements of the proposed DDS Tariff are reported below.

Channels Between Digital Cities

For Transmission Speed of:	Fixed Charge	Rate Per Airline Mile
2.4 Kbps	\$ 20.00/mo	\$ .40/mo. × mile
4.8 Kbps	40.00	.60
9.6 Kbps	60.00	.90
56 Kbps	125.00	4.00

Digital Access Lines In Digital City Serving Areas

Type I (≤ 5 miles from Telco office)

For Transmission Speed of:	Monthly Charge	Non-Recurring Charge
2.4 Kbps	\$ 65.00/mo	\$100.00
4.8 Kbps	85.00	100.00
9.6 Kbps	110.00	100.00
56 Kbps	200.00	150.00

Type II (> 5 miles from Telco office)

For Transmission Speed of:	Fixed Charge	Rate Per Airline Mile	Non-Recurring Charge
2.4 Kbps	$ 90.00/mo	$ .60/mo × mile	$100.00
4.8 Kbps	110.00	.90	100.00
9.6 Kbps	130.00	1.30	100.00
56 Kbps	250.00	6.00	150.00

Data Service Units

For Transmission Speed of:	Monthly Charge	Non-Recurring Charge
2.4 Kbps	$15.00/mo	$25.00
4.8 Kbps	15.00	25.00
9.6 Kbps	15.00	25.00
56 Kbps	20.00	25.00

DDS will be initially offered between 5 cities, and will be extended to include 96 cities in 1976. Since terminals outside the DDS cities require expensive analog interconnections, it is likely that the benefits of the DDS service will be felt much earlier in the backbone network, rather than in the local distributions.

Western Union, the other large common carrier, parallels AT&T in most of the analog offerings. In addition, Western Union offers to the data users a unique service known as Data Comm. The service is available in some 60 cities and is intended for users with a mixed set of low speed data requirements between two or three Data Comm cities. The low speed lines are time division multiplexed and demultiplexed by Western Union in the Data Comm offices. The transmission between cities is over voice grade lines.

Specialized carriers have the general connotation of offering analog and/or digital services of high quality, at low rate, between a limited number of cities, typically in high industrialization areas. The main features of some of the specialized carriers are described below.

MCI Telecommunications Corporation serves more than 20 cities stretching from New York to Washington, D.C., west to Chicago and south to Dallas and Houston. Data speeds range from 300 bps to 56 Kb/s, including an interesting 19.2 Kb/s offering. The rates are similar to the corresponding AT&T high density and bulk discount rates.

Southern Pacific Communication (SPC) offers nationwide data communications services from teletype up to 100 Kb/s speeds at very competitive rates. For example, the monthly cost for a voice grade New York to Los Angeles connection is $1,144, i.e., less than half the equivalent AT&T high density charge. These special rates apply between cities connected by SPC leased satellite channels. For other cities, the rates are still 10 to 20 percent less than the equivalent AT&T high density rates.

U.S. Transmission System, Inc. (USTS), one of the latest specialized carriers to receive FCC authorization, plans to establish a 1500 mile backbone microwave network from Houston to New York, with data speed offerings ranging from teletype to 960 Kb/s. A very diversified gamut of service offerings (part time usage; metered service; store and forward message switching; facsmile etc.) is being planned.

Western Telecommunication Inc. offers data services up to 50 Kb/s between four major western cities (Los Angeles, San Diego, Phoenix and Tucson), at rates somewhat lower than the AT&T equivalent. An agreement has recently been reached between Amersat, MCI and Western TCI, to extend the specialized service nationwide.

While the above mentioned carriers provide analog channels for data transmission, Datran is offering a truly digital service to compete with the DDS of AT&T. The service is available between about 10 major cities in the mid-west, and will be extended to the east and west coasts through an interconnect agreement with Southern Pacific Communications. Data speeds are 2.4, 4.8, 9.6 and 56 Kb/s. Ultra high speeds of 1.344 and 2.688 Mb/s will be available on a point-to-point basis. Datran rates are parallel to the DDS rates. Circuit availability better than 99.95 percent is promised, on a money back guarantee basis.

On the domestic satellite scene, several carriers are offering analog and digital channels with bandwidth up to 230 Kb/s (and even larger) between major U.S. cities, at extremely competitive rates. The basic rates offered by Western Union (excluding local loops) are shown below. Identical rates are offered by the other satellite carriers.

Service Route	Single Channel	Base Group (12 Channels)	Super Group (60 Channels)
New York-Los Angeles New York-San Francisco Atlanta-Los Angeles Atlanta-San Francisco Washington-Los Angeles Washington-San Francisco	$1,000	$10,800	$48,000
Chicago-Los Angeles Chicago-San Francisco Dallas-New York Dallas-Washington Dallas-Los Angeles Dallas-San Francisco	750	8,100	36,000
Chicago-Dallas Chicago-New York Chicago-Washington Atlanta-New York Atlanta-Dallas Atlanta-Chicago Atlanta-Washington	500	5,400	24,000
		(monthly rates)	

If not in the rates, the domestic satellite carriers differ from each other in number and location of ground stations; local distribution arrangements, large bandwidth offerings, channel quality and specialized digital services. The characteristics of the major satellite carriers are illustrated below.

American Satellite Corporation (Amersat) has stations in New York, Los Angeles and Dallas, with terrestrial connections to San Francisco, Chicago and Washington, D.C. Nationwide distribution is obtained through interconnect agreements with several specialized carriers. In addition, Amersat is considering the possibility of providing direct satellite connection to the customers, bypassing the AT&T local loop, with the application of advanced satellite technologies.

RCA Satcom offers service between Alaska, New York and San Francisco, with stations in Washington, D.C. and Los Angeles to be added in the near future, and 6 more stations to be added by 1976.

Western Union satellite service is provided between 7 major cities, directly or via the Western Union microwave network. Extensions to other cities are available at terrestrial private line rates. An interesting aspect of the Western Union service is the possibility of leasing an entire transponder (36 MHZ) for a rate ranging from 100 K$ to 180K$ per month, excluding ground stations. The customer can use his own ground stations, if he desires.

In summary, the new offerings both from AT&T and from specialized and satellite carriers provide a data service of better quality, higher reliability, greater flexibility and lower cost between a limited number of highly industrialized areas. However, these benefits are often lost when the end points of the connection are not within the urban areas covered by the service, since in those cases expensive, relatively unreliable and lower quality local loops must be used for the interconnection. Typically the adoption of a new data offering will lead to considerable dollar savings, better quality and higher flexibility in the backbone network, at the expense of a cost increase in the local distributions. Therefore, the match between the geographical distribution of user requirements and carrier stations plays a fundamental role in the selection of the appropriate data service.

Besides conventional and specialized carriers, the user will have yet another alternative to consider, namely the data service provided by the value added carriers. Since the common connotation of such carriers is the adoption of the relatively new packet-switching technique, a brief description of such technique is provided in the next section.

HOW PACKET-SWITCHING MOVES DATA

Packet-switching technology is a spinoff from the development of ARPANET, a distributed network which interconnects more than 40 research installations of the Advanced Research Projects Agency (ARPA). Operational since the summer of 1971, the network was developed with more than $10 million of government funds to explore networkt echnology and gave researchers at ARPA-sponsored centers the facility to share each other's programs, services and data bases.

For ARPANET, researchers developed the distributed packet-switching concept to achieve lower costs, higher speeds, greater reliability and greater flexibility than had been realized before in data networks. Virtually error-free communications are possible from almost any known terminal type to any of a variety of computers, as well as between computers. The advantages of distributed packet-switching both for terminal-to-computer and computer-to-computer communications make the system one of the most significant recent contributions to the field of data communications.

Packet-switched networks use leased lines as transmission links and minicomputers for store-and-forward message switching and network control. Many of the network's desirable characteristics result from the use of adaptive routing, where the path through the network between any two points is not chosen in advance but is a dynamic function of conditions in the network at any time. With its ability to reallocate its resources as needed, the network overcomes adverse effects of temporary congestion and failed links or switches. Packet-switched networks utilize a powerful error-control scheme, and an undetected error can be expected to occur only once every few years. Message delivery to the addressee is confirmed with an acknowledgment message returned to the sender.

Each switch in the network functions as a "local" network manager, deriving its management information from the network. This function is implemented through a Network Control Center (NCC) which appears to the network as another data processing computer facility. The NCC automatically collects comprehensive status reports from all switches and provides for extremely effective "global" network management.

A typical example of packet-switched network configuration is offered by ARPANET. In the network, each user computer is called a host. User terminals and host computers are connected to the network through two types of minicomputers: an Interface Message Processor (IMP), which interfaces one or more host computers with the network; and a Terminal Interface Message Processor (TIP), which performs the functions of an IMP and also interconnects the network directly with up to 63 user terminals or consoles. Serving as a simple host, a TIP converts the characteristics of diverse types of terminals to a network standard. IMP's provide the standard interface for each host computer, and perform all communications functions.

In ARPANET, the IMP's and TIP's are connected by leased communication lines and may use a wide range of communication channel data rates up to 230 kbps. Each IMP handles its communications tasks completely independently of the host computers, and the network operates under a distributed control scheme, where each IMP and TIP makes its own decisions as to control of communications with its host and routing of message traffic through the networks. To send a message to another host, the computer precedes the text of its message with an address and delivers it to its local IMP; this mini computer dynamically determines the best route provides error control, and notifies the sender of its receipt. TIP's perform the same function for terminals.

When a message is ready for transmission, the originating IMP or TIP divides the message into a set of one or more packets, each with appropriate header information. Each packet makes its way independently through the network to the destination IMP or TIP, where the packets are reassembled into the original message and then transferred to the destination host or terminal. Since parts of a message may take different paths through the network, unauthorized access to a transmission link allows only partial interception of the messages, so that packet-switching networks provide enhanced security. Also, in a packet-switched network whose facilities are distributed across the United States, the cost of sending data between two distant points is approximately the same as the cost of sending data between two relatively close points.

Essentially, packet-switching is a specialized form of store-and-forward message switching. However, it differs significantly from both conventional message-switching and the circuit-switching techniques employed in the public telephone network.

In a circuit-switched network, the entire transmission path between sender and receiver is chosen in advance, and for the period of the call, network resources are allocated to the exclusive use of the conversation, whether or not there is any conversation or data being sent. In a conventional message-switching network, data messages are sent along a predetermined path from sender to receiver; however, messages can be temporarily stored at intermediate relay points. This storage capability means that network circuits are not allocated in advance, but as they become available. By delaying the delivery of a message from sender to receiver, message-switching effectively spreads peak demand for service over time and thus more efficiently utilizes network facilities.

VALUE ADDED NETWORKS (VAN's)

Value added networks are communication service companies which lease transmission facilities from common or specialized carriers and resell communication services not available from the original carrier. One of the features commonly offered is packet-switching, with service comparable to that in the ARPANET. FCC approval has been granted to GRAPHNET, Packet Communications, Inc., and Telenet. The introduction of VAN's services may be quite rapid since they do not undertake the construction of new transmission lines.

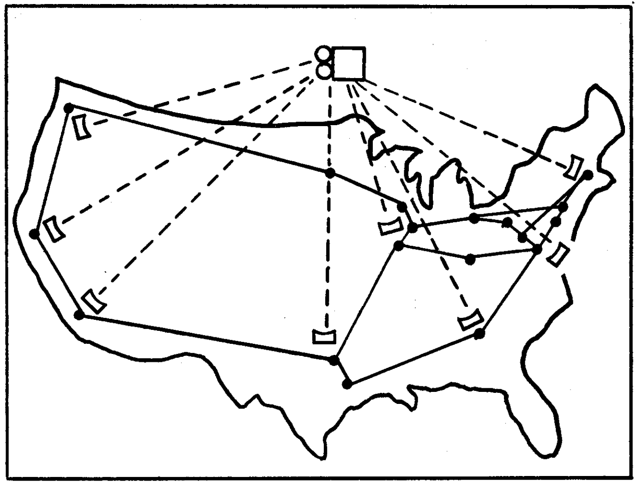


Figure 1—Proposed Telenet configuration

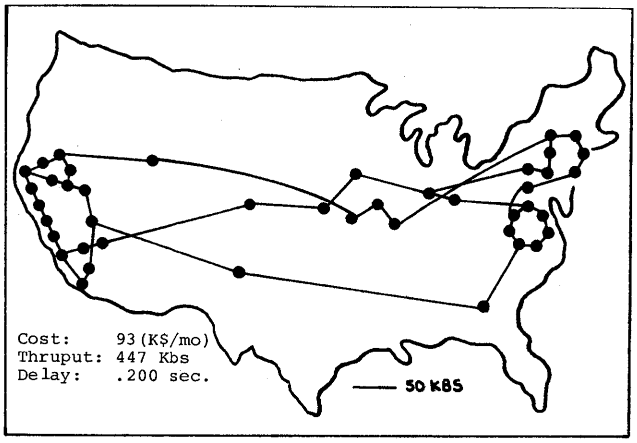


Figure 2—Nationwide computer network

Typical value added services deriving from the packet-switched implementation are: automatic terminal speed recognition and conversion; code translation; powerful error detection and correction; high network availability; easy access to distributed resources.

Network charge consists of two components:

1. The charge for the usage of dedicated or dial-up ports at the packet-switching centers; and
2. The charge for the volume of data transmitted (typically independent of distance travelled).

In addition to VAN charges, the user must pay for the lines (dedicated or dial-up) from terminals to VAN service centers. If there is no good match between user locations and VAN locations, the local access charge might actually exceed the direct VAN charge, as shown in an application at the end of this paper.

PCI was the first VAN to obtain FCC authorization in November 1973. The initial PCI network will connect 18 major US cities via terrestrial, wideband lines. Future plans call for the extension of the service to 40 cities.

Telenet also plans an 18-city network, which will use terrestrial as well as satellite links. The proposed configuration is shown in Figure 1. The tariff filed by Telenet is reported below:

Packet charge	:	$ 1.25/Kilopacket
Dedicated port charge	:	$ 50/mo (up to 9.6Kb/s) $100/mo (50Kb/s)
Dial-in port charge	:	0-1800bps $1.00/hr. 2400bps $2.00/hr. 4800bps $3.5/hr.

It is anticipated that value added services will have a profound impact on network design strategy, especially for the small and medium data communications user. In fact, such users might find it advantageous to replace the traditional private backbone network with the shared use of a

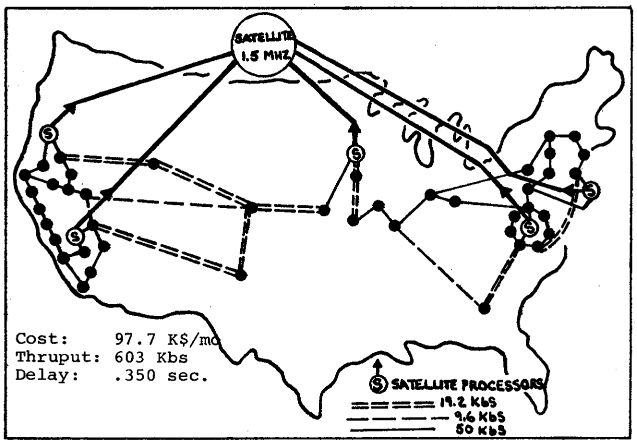


Figure 3—Satellite upgraded configuration

VAN, for better quality, flexibility growth capability and, possibly, lower cost.

SELECTING THE BEST ALTERNATIVE

The new services offer potential benefits that are extremely attractive, and certainly must be considered by the cost-conscious communications user. It must be remembered, however, that the effective use of such services often places new constraints on overall network design. A fundamental restriction is represented by the fact that the services are generally available only in a limited number of urban areas, so that the user with geographically sparse requirements often loses the cost and quality benefits when interconnecting terminals outside the cities in which the service is available.

Therefore, it is imperative that users explore the largest possible number of alternatives. But, in performing this evaluation, they must identify and accurately analyze all network cost components (backbone, local distribution, communication hardware and software, etc.) and, if necessary, reoptimize network topology and strategy for each alternative. Furthermore, the comparison cannot be limited to cost and performance criteria relative to the present communications needs, but must be extended to consider also growth capability and flexibility in meeting future requirements.

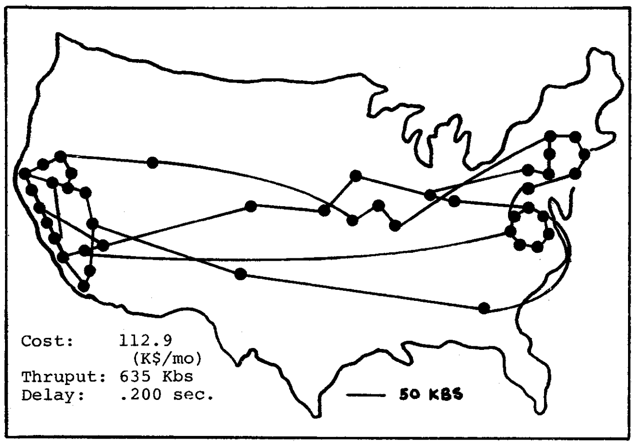


Figure 4—Terrestrial upgraded configuration without satellite

Two examples of evaluation of different service alternatives for network design are reported here. The first example is relative to the expansion of a large computer network using terrestrial or satellite links. The second example compares in-house backbone implementation versus rental of VAN services for a medium-sized, terminal oriented network.

Under contract with the Advanced Research Project Agency, we recently evaluated the cost-effectiveness of using satellite services in order to upgrade the capacity of ARPANET, the large nationwide computer network whose continental links are now implemented exclusively with wideband terrestrial channels.[1,2] The satellite alternative consisted of a satellite channel of 1.5 MHz bandwidth made available at the vendor ground stations at a yearly rate of $100,000. This charge does not include the cost of local loops.[3]

The study was carried out by optimally upgrading network capacity to meet a 50 percent increase in traffic, using either the terrestrial or the satellite alternative. The results, illustrated in Figures 2, 3, and 4, show that the satellite alternative is more cost-effective, and leads to total cost savings on the order of 10 percent. It might be noticed that a major reoptimization of the terrestrial network was necessary, in order to take full advantage of the low cost satellite bandwidth.

The advantages of the satellite solution are not limited to cost savings. In fact, the satellite channel offers more flexibility in adjusting to changes in traffic requirements (this property is inherent in the satellite multiple access channel) and better growth capability (the user can provide his own ground stations and lease more satellite bandwidth, with formidable volume discounts).

In another example, we compare two alternative design strategies—private backbone network and Telenet—for the implementation of a medium-sized network with a few

TABLE I—In-House and Telenet Costs

IN-HOUSE ALTERNATIVE	
Total Network Cost	$42,000/mo.
TELENET ALTERNATIVE	
Data Transmission Charge	$ 1,300/mo.
Port Charge	2,600
Local Access Cost from Terminals to Telenet sites	21,500
Cost of Direct Terminal to Host Connections	800
TOTAL NETWORK COST	$26,200/mo.

hundred terminals sparsely distributed across the nation and the host computer located on the East Coast.

In the in-house network case, TDMX devices are strategically located across the nation, and are connected to the host via private trunks. Terminals are connected to the nearest TDMX (or to the host) via dedicated or dial-up line, depending on connect time and frequency of usage.

In the Telenet alternative, terminals are connected to the nearest packet-switching station, or to the host (when more economical).

The results of the evaluation are reported in Table I. Private and dial-up line charges were computed according to the current AT&T tariffs; Telenet charges were determined according to the tariffs shown earlier. The results indicate that the Telenet solution is much more cost-effective than the in-house solution. Furthermore, the use of Telenet offers better terminal growth capability, and better flexibility to changes in traffic pattern and, possibly, to distributed host and data base implementations.

CONCLUSIONS

The future will see a rapid growth of conventional and specialized data communications offerings, with tariffs subject to frequent changes, mainly because of the competition between carriers and the development of new techniques.

The cost conscious user must be prepared to react to this dynamic communications market. In particular, he must be prepared to explore a large number of alternatives during the network implementation phase, and must be ready to reconfigure his network more often than before, in order to take advantage of rapidly changing cost and quality of service.

The evaluation of different alternatives must be very accurate and comprehensive, to identify and appraise all network cost components (communications hardware and software, local distribution lines, backbone trunks, etc.). Sophisticated network design and evaluation tools become an absolute necessity, especially for networks of considerable size.[4]

The selection of the best alternative is not based uniquely on cost. In fact, in a competitive environment, it is likely that the same channel bandwidth between the same points will be offered at similar cost by all carriers. However, one carrier may differ from another for channel type and quality, availability of service, number of cities served (or planned for service), provision for interconnection with other carriers, etc. In evaluating VAN's, for example, type and quality of the value added service, rather than the mere data transmission cost might be the overriding consideration.

Finally, the user, in making his decision, must carefully analyze his present and future communications requirements, and for each alternative, determine not only the cost of satisfying his present needs, but also the growth capability and flexibility in meeting future needs.

ACKNOWLEDGMENTS

This work was supported by the Advanced Research Project Agency of the Department of Defense under Contract No. DAHC-15-73-C0135.

REFERENCES

1. Roberts, L. G. and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *AFIPS Conference Proceedings*, 36, pp. 543-599, SJCC, Atlantic City, New Jersey, 1970.
2. Frank, H., I. T. Frisch, and W. Chou, "Topological Considerations in the Design of the ARPA Computer Network," *AFIPS Conference Proceedings*, 36, pp. 581-587, SJCC, Atlantic City, New Jersey, 1970.
3. Network Analysis Corporation, "Practical Impact of Recent Computer Advances on the Analysis and Design of Large Scale Networks," *Second Semiannual Technical Report*, December, 1973, available from the Defense Documentation Center, Arlington, Va.
4. Gerla, M., "New Line Tariffs and Their Impact on Communications Network Design," *Proceedings of the National Computer Conference*, Chicago, Illinois, May, 1974.

Speech transmission in packet-switched store-and-forward networks*

by JAMES W. FORGIE
M.I.T. Lincoln Laboratory
Lexington, Massachusetts

INTRODUCTION

The past few years have seen a widespread and growing application of packet-switched store-and-forward networks for data communication between geographically separated computer installations.[1,2] Such networks can provide connections between facilities with the many desirable properties of dedicated communication lines but at reduced costs which result from the time-sharing of the actual lines among many customers. Barring the failure of local terminal equipment, such a network provides connections among its customers which implicitly exist regardless of the load being placed on the network. However, the network characteristics seen by an individual customer will vary with overall network traffic, since the network will tend to deliver messages less frequently and with increased delay (possibly beyond the point of usefulness) as load increases. This property of packet-switched networks—guaranteed connections with variable throughput and delay—is just the reverse of that exhibited by circuit-switched networks such as the telephone system. When circuit-switched systems are heavily loaded, delay may be experienced in making a connection, but once it is established, its throughput and delay will not vary with other system activity.

Interest in transmitting speech in packet-switched networks grows out of the value of the guaranteed connection offered by such a network. The advantage of a guaranteed connection is obvious in many governmental and commercial situations, and such connections are often realized today by dedicating leased circuits to this function. If speech communication could be handled effectively with packet-switching techniques, then some economies could be expected by combining speech with data traffic and using the same network to provide both services.

Other benefits could be expected from handling speech in digital form as would be required to be compatible with the data in a combined network. Digital transmission is inherently insensitive to noise, crosstalk, and distortion. Encoding to insure privacy and security is greatly simplified. By treating speech as data and introducing it into computers on the network, a number of interesting new possibilities are opened up for supporting conferencing and handling spoken messages on a non-real-time basis.

The purpose of this paper is to discuss the technical problems which must be overcome if satisfactory speech communication is to be achieved in a packet-switched network. The paper is semi-tutorial in nature and attempts to provide some background for those who may be unfamiliar with the nature of the speech communication process and/or the characteristics of packet-switched networks. In addition to a statement of the problem, the paper concludes with some requirements which the author feels must be met by networks if they are to support speech communication.

CHARACTERISTICS OF THE SPEECH DATA STREAM

In order to transmit a speech signal through a packet-switched store-and-forward network, it is necessary to convert it to an appropriate digital form. Many techniques are available to achieve this transformation, and they vary greatly in data rate, hardware complexity, and quality of the output speech. If high-fidelity reproduction of the speech waveform were to be required, and conventional analog-to-digital conversion techniques were used, the resulting PCM (pulse code mudulation) representation of the signal would require the communication system to handle a data rate of about 250,000 bps (bits per second). If "telephone" quality speech were adequate, and the same techniques were used, only about 50,000 bps of capacity would be needed. PCM techniques imply high required data rates, low complexity and cost, and good quality.

Since high data rates mean high communication cost, much effort has been directed to the exploration of schemes to handle speech satisfactorily at lower data rates. In the years since Dudley's invention of the vocoder,[3] activity in the speech bandwidth compression area has followed a somewhat cyclical pattern. At the moment research in the area is relatively intense, spurred by recent theoretical advances, the availability of high-speed processors capable of simulating vocoders in real time, and the continuing downward trend in component costs for digital hardware. Space limitations preclude even a

* This work was sponsored by the Advanced Research Projects Agency of the Department of Defense.

cursory review of the techniques presently being investigated. The interested reader can find in References 4 through 8 both overviews[4,5] and more detailed discussions of particular techniques. References 5 and 6 also include recordings which allow subjective evaluation of the processed speech.

Broadly speaking, two approaches to bandwidth compression have been pursued, each with many variations. The first might be called a waveform coding approach. Here an attempt is made to reconstruct a good replica of the original speech waveform by taking advantage of the fact that successive samples of the input signal are not independent of each other, as would be allowed by the PCM representation, and that consequently fewer bits are needed to represent the range of possibilities for the successor to a known sample. The current state of the waveform-coding art offers devices of modest cost and complexity which can produce good quality speech at data rates of 16,000 to 24,000 bps.

The second, or vocoder approach, abandons the concept of re-creating the original waveform. Instead, the input is analyzed in terms of a model of the speech process. The results of the analysis are transmitted to the receiver where a speech-like signal is synthesized. Ideally, the synthesized signal will sound very much like the original speech.

Vocoders have been available for some time which operate in the range of 2400 to 9600 bps, and experimental vocoders have been demonstrated at 1200 bps. While intelligibility tests suggest that vocoders operating toward the lower end of the range should be adequate for speech communication, they have not been widely used. The market for such vocoders has been small because of high cost and the somewhat unnatural quality of the output speech. The adequacy of the speech quality from vocoders has often depended on the characteristics of the talker's voice so that a particular vocoder may seem quite satisfactory for some voices and very bad for others. A substantial fraction of current research effort is being directed toward the exploration of linear prediction techniques[6,7] which promise to give improved quality and talker independence in the 2000 to 4000 bps range.

Since most speech transmission has involved circuits of constant bandwidth, it has become the custom to think of a speech signal as requiring a constant data rate equal to the peak rate. However, for the purposes of packet-switched communications it is useful to consider a speech signal as a variable-rate bit stream. Obviously, no bits at all need to be transmitted when the talker is silent, either pausing to think or to wait for the other party in a conversation to finish talking. The statistics of this on-off aspect of the speech data stream have been investigated[9] and used to advantage in the TASI* system to at least double the number of conversations which a group of speech channels can handle. Unfortunately, while the most frequently occurring "talkspurts" have durations of less than half a second, many go on for 10 seconds or more, and the statistics are such that the communication system designer cannot take much advantage of the potential saving in channel capacity unless the system capacity is large enough to handle 20 to 50 simultaneous conversations.[10]

Further reductions in average data rate are possible by taking advantage of the fact that during certain speech sounds the character of the sound changes much more slowly than its maximum rate.[11] In order to make use of this type of variable data rate, buffering is required at both ends of the transmission link. Since buffering is already required to cope with the variable bandwidth of a packet-switched network, it would appear desirable to take advantage of this fact by using a variable-rate vocoding technique. Since work on such techniques is still at an early research stage and cost/benefit ratios are not yet available, it is premature to assume that such devices will prove to have practical advantages. It is likely, however, that speech transmission in a network will take advantage of the on-off aspects of the speech signal.

In comparing waveform-coding devices with vocoders, it appears safe to assume that a vocoder will be more expensive than a waveform-coder giving comparable speech quality. A vocoder is also likely to be more sensitive to errors in the data stream. On the other hand, both the communication costs and the delay introduced into conversations will be greater for the waveform-coder with its expected factor of five to ten higher data rate. The choice of the technique to be used in a particular situation will depend on overall costs, delay characteristics of the network, and subjective quality judgments. Present knowledge does not suggest an obvious choice for general use in packet-switched networks.It is therefore reasonable to expect that both high (16,000-24,000 bps) and low (2000-4000 bps) rate devices will find application in such nets.

CHARACTERISTICS OF PACKET-SWITCHED NETWORKS

In a packet-switched communication network, customers are provided with ports which accept and emit data streams made up of entities called messages and associated identifiers. The identifiers must contain enough information to specify the destination to which the message is to be sent, and they may contain additional information such as requests for special handling by the network. A message given to the net is not generally forwarded immediately. It must wait until an opportunity arises to transmit it along a shared wire line, radio channel, satellite link, or whatever to another node in the net. There it is likely to wait again for an opportunity for transmission to yet another node. Eventually it will arrive at a node which can deliver it to its intended destination. The number of nodes through which a message will pass cannot be less than some minimum determined by the to-

* TASI[10], an abbreviation for Time Assigned Speech Interpolation, is a high-speed transmission and switching system used in some long-distance telephone systems to interpolate additional talkers into the idle channel time present in telephone conversations.

pology of the net, but it may be larger if the network makes use of alternate routing to avoid trouble at some node or communication channels which are congested or have failed.

In order to provide reliable, error-free communication in the presence of errors in the communication links, the network will add extra bits to a message to allow error detection or correction. To avoid propagating errors in the net, the network control algorithm is likely to require that a node receive an entire message and check its integrity before starting to send it to the next node on its route. As a result the message will experience some inherent delay in traversing the net, independent of the availability of communication channel capacity and other traffic in the net which may cause further delay. The minimum network transit time may be written

$$T_{\min}=\sum_{i=1}^{n-1}(L/C_i+P_i+S_i+R_{i+1})$$

where n is the number of nodes through which the message would pass on a best route through the net,

L is the length of the message (in bits) within the net (i.e., L includes overhead bits for destination codes, error detection, line synchronization, etc.),

C_i is the channel capacity (in bits per second) of the communication link between node$_i$ and node$_{i+1}$,

P_i is the propagation time for the communication link between node$_i$ and node$_{i+1}$,

S_i is the processing time associated with transmission at node$_i$,

and R_i is the processing time associated with reception at node$_i$.

The actual total transit time experienced by a message will be longer than $T_{\min}$ by the time spent waiting at the nodes due to other traffic and the time required to transfer the message between the network ports and the source and destination customers. Additional delay may be introduced by a need to retransmit the message because of communication link errors or buffer overflow problems at one or more nodes.

Since messages can follow each other through the net unless prohibited by network control algorithms, the minimum time to deliver k equal length messages is not $kT_{\min}$ but

$$T_{\min}+(k-1)\max_{i=1,n-1}(L/C_i+S_i).$$

This expression denotes the time for the first message to cross the net plus the time for the succeeding $k-1$ messages to pass through the node for which the sum of transmission handling time and link transmission time is maximum.[12] Since this expression is generally less than $kT_{\min}$, it is desirable for a customer who wants fast delivery to break long messages into sequences of shorter messages. For example, let us assume an equal channel capacity of 1 Mbps for all links, a route involving 10 nodes, an overhead of 200 bits per message, handling time of 0.5 msec per node, and an end-to-end propagation time of 20 msec. Under these conditions, a 20,000 bit message would require a minimum of 227 msec to traverse the net, but a sequence of 20 messages each 1000 bits long could get through the net in only 69.2 msec. This advantage of short messages is even greater when the transfer time between the network and the source and destination customers is taken into account.

The relative advantage of shorter messages holds in the presence of other traffic, but since in that case the network is interleaving messages from many streams, maximum benefit can be obtained only if all messages are kept short. This goal can be accomplished by limiting the maximum length of messages which the network will accept or by using packetization techniques. In the latter case, the network arbitrarily chops long messages into packets of length no greater than the desired maximum. Packets are then handled within the net as independent messages each with its own destination code, error checking, etc. On arrival at the destination node, the packets are reassembled into messages before being delivered to the destination customer. Actually, the entities sent from node to node are called "packets" by the designers of packet-switched networks whether the entities are messages of limited size or pieces of longer messages, hence the name "packet-switching". The choice of an optimum maximum packet length depends on many network design parameters, and its discussion is beyond the scope of this paper. It is important to note, however, that since delay is a critical parameter in voice communication, it is likely that a network will have to enforce a low limit on message lengths or make use of packetization with a short maximum packet, if it is to handle speech satisfactorily.

SPEECH IN PACKET-SWITCHED NETWORKS

Barring catastrophic failure or gross overloading, the only aspect of store-and-forward networks which poses a problem for speech communication is the delay which they introduce into the speech data stream. While it is true that under overload conditions the average throughput of a network may fall below the required data rate for speech transmission, a speech receiver can wait for enough data to accumulate before beginning to reconstitute the output speech to avoid destroying the intelligibility of the speech. The delays which would result in such a situation could be very long and might prevent normal conversational use of the network, but communication would still be possible. Such overload conditions would hopefully be rare in a network designed and sized to serve a community of speech customers. However, packet-switched store-and-forward networks can be expected to introduce delays into the speech data stream which could cause problems even under normal conditions.

Although delay has no effect on the intelligibility or naturalness of a speech signal, when it is introduced into a conversational situation, it becomes readily detectable and can have disruptive effects on the conversation. With the anticipated use of stationary satellites for speech communication, experiments[13,14] were undertaken to evaluate the effects of delays of the order of 0.6 seconds which would be expected in the round-trip time to such satellites. The results showed that the effects of delays of this amount or more were largely of a psychological nature. Telephone conversations normally involve frequent interaction between the participants even though one person may be doing most of the talking for an extended period. When the reinforcing feedback of an expected "yes", "really?", or whatever is delayed, the talker gets the feeling that the other party is not paying proper attention, and he tends to become irritated. Similarly, when the other party tries to interrupt the speaker, he becomes annoyed because the speaker appears to be ignoring his attempt to interrupt.

The nature of the problem posed by delay is such that one would expect people to adjust their behavior to cope with it, and experiments[15,16] have shown that such adjustment does occur. Speech patterns change, with the most noticeable effect being a tendency to cling to the role of talker for longer periods. These results assume that any delayed echo of the speaker's voice has been eliminated. Such a delayed echo would result from any crosstalk (electrical or acoustical) between the signal being received at the far end of the communication system and the signal being transmitted from there. Crosstalk of this kind occurs in the ordinary 2-wire local telephone system, and it is dealt with in long distance telephony by the use of special echo-suppressor circuits. A delayed echo is very disturbing, and it can seriously interfere with a person's ability to speak coherently and intelligibly. It is important to avoid such echoes in a packet-switched speech system, but to do so poses no consequential problems.

The variability of the delay in a store-and-forward network also poses problems for speech communication. For example, the transmitter may chop the input speech into chunks of equal length and give the corresponding messages to the net at equal time intervals. When they arrive at the receiver, the time between messages is no longer likely to be uniform but will generally exhibit considerable variation, and the receiver must take appropriate action to compensate for this jitter. Depending on the network control algorithms, the messages may even arrive in a different order than the one in which they were sent. In that case a sequence number or some equivalent information will have to be added to the messages to allow the receiver to reproduce the speech in the proper sequence.

When the receiver is reconstituting the speech from the message stream, and a message has been abnormally delayed in the net, a point may be reached where all the available messages have been used up. If this point corresponds to a pause in the input speech, all will be well. Otherwise a gap or "glitch" will be introduced into the output speech. The glitch may be left as a silent gap, or if a vocoder is being used to synthesize the output speech, the last vocoder frame may be repeated to fill the glitch. This latter procedure has the effect of stretching the speech and could be a good technique for handling short glitches. Long glitches are likely to be best left as silent gaps. Unfortunately, it will often be the case that the receiver will not have any information about the expected duration of the glitch. Only in the situation where messages are delivered out of order and a successor to a late or missing message has been received will an upper bound on the glitch duration be available. In any event, the occurrence of glitches will tend to increase the duration of output talkspurts in relation to the corresponding inputs. This effect requires the receiver to make corresponding adjustments in the length of the pauses between talkspurts to avoid a situation where the output gets farther and farther behind the input.

Unlike delay, which has a primarily psychological effect on a conversation, glitches can effect the intelligibility of the output speech. Experiments to assess the subjective effects of the sort of glitches to be expected with packet-switched speech and the various schemes for dealing with them have not yet been carried out. It seems reasonable to expect that the glitch rate will have to be kept low for satisfactory speech communication in a packet-switched system. Observations with the TASI system,[10] which can produce similar glitches (but only at the start of a talkspurt), show that a glitch rate of 0.5 per cent is readily detectable, and two percent is disturbing to the continuity of a conversation.

In order to keep the glitch probability low, the receiver will have to introduce some additional delay in the speech stream to smooth the jitter in message arrival times. The magnitude of the smoothing delay required to achieve a glitch probability less than some given value will depend on the dispersion of network transit times. This dispersion is caused by contention for resources among the messages being handled by the net. Network control algorithms have some influence on this dispersion, and if speech is to be handled satisfactorily, they should be designed both to minimize the mean dispersion and to limit the extremes. Of course, the true extremes, caused by transmission link or node failures, cannot be controlled, but these should not occur frequently enough to be troublesome. Similarly, the speech receiver should be designed to keep the smoothing delay as short as conditions allow by adjusting it periodically on the basis of the observed dispersion of message arrival times in its data stream.

Since the speech data stream has predictable properties and stringent delay requirements, it is likely that a network designed to handle speech as well as data would give different service to the two kinds of traffic. To minimize delay, speech messages would be given priority on the communication links. In order to handle the resulting preemption of data traffic without requiring retransmission, it would be necessary to reserve appropriate buffer space at the nodes. Fortunately, the size of such buffer space is determined by the relatively predictable speech traffic. To provide the speech receiver with the maximum available information, speech messages would be delivered as they arrive at the destination node without regard to the order in which they were transmitted. If the receiver chooses to operate in a mode in which it proceeds without waiting for abnormally delayed messages, it would be appropriate for the network to discard stale messages without wasting further effort on attempts to deliver them. This mode of behavior should be under customer control since there may be a need to record the speech for later use, in which case the delayed messages could be important.

While there is no need for the confirmation of the successful delivery of individual messages in a speech stream, it would be desirable for the network to send appropriate messages to the transmitter in the event that failure to make a timely delivery has occurred. Such failures could occur because of momentary or prolonged overload conditions or because of communication link or node failures. The transmitter could indicate the fact that trouble had been detected to the talker who could then take appropriate action.

CONCLUSIONS AND OBSERVATIONS

The principal effect noticed by users of a packet-switched voice communication system would be the delay introduced into their conversations. While such delays are readily detected by people accustomed to communication systems without appreciable delay, experience has shown that people can adapt to delays of the order of 0.5 to 1.0 second without great difficulty. Packet-switched networks can be designed and implemented using current technology which can keep delays within that range. Such networks would make use of appropriate combinations of communication link bandwidths and network topologies, and they would keep message lengths short within the net either by setting limits on allowable message lengths or adopting packetization techniques. Care would have to be taken in the design of the network control algorithms to minimize average delay and to control the dispersion of delays seen by speech messages.

This paper has not addressed the many economic issues involved in evaluating the prospects for packet-switched networks capable of giving satisfactory service for both speech and data. These issues will play an important role in deciding whether such networks are actually built in the future. Without going into a detailed analysis of costs it is hazardous to predict whether a network designed to have delay characteristics suitable for voice use would have higher or lower per unit message costs than one designed solely for data applications. It is clear that using more and higher capacity communication links to achieve better delay characteristics would result in higher overall system costs. Similarly, the extra processing power required at the network nodes would add to overall costs. On the other hand, the traffic handling capacity of the faster net would be larger, and non-linear communication tariffs and processor costs might well result in lower per unit costs.

The potential advantage posed by low data rate speech coding devices over high rate ones can be used in a network to achieve higher total capacity, lower average delay, or some combination of the two. Economic considerations favor using any such advantage to increase capacity and thus reduce the cost per conversation. The same arguments would allow average delay to rise toward the upper end of the acceptable range.

This paper has been concerned with evaluating the prospects for achieving satisfactory voice communication in packet-switched networks. While attention has been focused on networks using pure packet-switching techniques, it is not the intent of the paper to argue that such networks are the only kind to be considered for handling mixed voice and data traffic. Other schemes have been proposed for handling such traffic in a digital communication system by combining line-switching and packet-switching techniques.[17] Straightforward use of line switching for voice traffic would avoid the delay problems discussed in this paper, but it would also lose the benefits of the guaranteed connection property inherent in packet switching. However, it may well be possible to design a mixed system which could retain the advantages of both techniques.

ACKNOWLEDGMENTS

The author wishes to acknowledge the efforts of his colleagues L. F. Mondshein, J. Tierney, and C. J. Weinstein in providing useful reference material, discussion of the issues, and comments on the paper. Helpful comments were also provided by the reviewers.

REFERENCES

1. Roberts, L. and B. Wessler, "Computer Network Development to Achieve Resource Sharing," *AFIPS Conference Proc.*, Vol. 36, pp. 543-549, 1970 Spring Joint Computer Conference.
2. Pouzin, L., "Presentation and Major Design Aspects of the Cyclades Computer Network," *Proc. 3rd Data Communications Symposium*, pp. 80-88, November 1973.
3. Dudley, H., "The Vocoder," *Bell Lab. Record 17*, pp. 122-126, 1939.
4. Flanagan, J. L., *Speech Analysis Synthesis and Perception*, second edition, New York, Springer-Verlag 1972, pp. 321-406.
5. Bayless, J. W., S. J. Campanella, and A. J. Goldberg, "Voice Signals: Bit-by-Bit" *IEEE Spectrum*, October 1973, pp. 28-34.
6. Atal, B. S. and S. L. Hanauer, "Speech Analysis and Synthesis by Linear Prediction of the Speech Wave" *J. Acoust. Soc. Am. 50*, pp. 637-655, 1971.
7. Markel, J. D., A. H. Gray, Jr., and H. Wakita, "Linear Prediction of Speech-Theory and Practice," *SCRL Monograph No. 10*, Speech Communication Research Lab., Santa Barbara, Calif., 1973.
8. Abate, J. E., "Linear and Adaptive Delta Modulation," *Proc. IEEE 55*, pp. 298-308, 1967.
9. Norwine, A. C. and O. J. Murphy, "Characteristic Time Intervals in Telephonic Conversation," *Bell System Tech. J. 17*, pp. 281-291, 1938.
10. Bullington, K. and J. M. Fraser, "Engineering Aspects of TASI," *Bell System Tech. J. 38*, pp. 353-364, 1959.
11. McLarnon, E., J. N. Holmes, and M. W. Judd, *Experiments with a Variable-Frame-Rate Coding Scheme Applied to Formant Synthesizer Control Signals*, Preprints of the Speech Communication Seminar, KTH, Stockholm, Sweden, pp. 71-79, 1974.
12. McQuillan, J. M., *Adaptive Routing Algorithm for Distributed Computer Networks*, Bolt Beranek and Newman Inc., Report No. 2831, Cambridge, Mass., p. 90, May 1974.
13. Emling, J. W. and D. Mitchell, "The Effects of Time Delay and Echoes on Telephone Conversations," *Bell System Tech. J. 42*, pp. 2869-2891, 1963.

14. Riesz, R. R. and E. T. Klemmer, "Subjective Evaluation of Delay and Echo Suppressors in Telephone Communication," *Bell System Tech. J. 42,* pp. 2919-2941, 1963.
15. Klemmer, E. T., "Subjective Evaluation of Transmission Delay in Telephone Conversations," *Bell System Tech. J. 46,* pp. 1141-1147, 1967.
16. Krauss, P. M. and P. D. Bricker, "Effects of Transmission Delay and Access Delay on the Efficiency of Verbal Communication," *J. Acoust. Soc. Am. 41,* pp. 286-292, 1967.
17. Zafiropulo, P., "Flexible Multiplexing for Networks Supporting Line-Switched and Packet-Switched Data Traffic," *Proc. 2nd Int. Conf. on Computer Communications,* pp. 517-523, 1974.

Dynamic control schemes for a packet switched multi-access broadcast channel*

by SIMON S. LAM
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

and

LEONARD KLEINROCK
University of California
Los Angeles, California

INTRODUCTION

Domestic satellites are emerging as an exciting alternative to satisfying the communications requirements of data users, providing both flexibility and economy. Two attributes of satellites are especially advantageous for the transmission of data in large geographically distributed computer networks. They are (i) the availability of wide transmission bandwidths over long distances and (ii) the multi-access broadcast capability inherent in radio communications which permits transmission to, and reception from, all points in a satellite connected network. These considerations also apply (on a smaller geographical scale) to the use of ground radio channels in a terminal access computer-communication network exemplified by the ALOHA System at the University of Hawaii.[1]

The random access scheme of the ALOHA System has inspired a number of packet switching techniques which permit the sharing of a high-speed multi-access broadcast channel by a large population of channel users.[2–8] Such packet switched radio systems (both satellite and ground radio) have a number of advantages over conventional wire communication techniques for computer communications, such as: the elimination of complex topological design and routing problems in large networks, the possibility of mobile users, the cost reduction over long distances and the increased flexibility for system reconfiguration and upgrading. Another attractive feature is that in these systems each user is merely represented by an ID number. Thus, the number of active users is bounded only by the channel capacity and there is no limitation to the number of inactive (but potentially active) users beyond that of a finite address space. Moreover, measurement studies have shown that interactive computer data traffic tends to be bursty.[9] A single high-speed radio channel permits the total demand of a large population of bursty users to be statistically averaged at the channel. Furthermore, each user transmits data at the full wideband data rate of the radio channel. Such efficient sharing and wideband transmission are in general not possible in a geographically distributed computer-communication network using wire communications.

Of interest in this paper is the slotted ALOHA random access scheme.[3,4,7,10–13] A slotted ALOHA channel multi-accessed by a large number of users has been shown to exhibit unstable behavior, i.e., the system may drift into an undesirable saturation state with a virtually zero probability of transmission success as a result of repeated user conflicts.[4,7,10–12,14] In this paper, a model is first presented for a slotted ALOHA channel supporting input from a large population of bursty users; the data rate of each channel user is assumed to be much less than the channel transmission rate. The underlying concepts of channel stability are then introduced. A dynamic channel control model is next presented and four dynamic channel control algorithms are given. The performance of these algorithms are tested through simulation and compared to analytic results previously obtained.[7,13] We conclude that these algorithms are capable of preventing the occurrence of channel saturation under temporary channel overload conditions and at the same time achieving a level of channel performance close to the theoretical optimum.

The slotted ALOHA model here is similar to one previously studied by Metcalfe through a steady-state analysis.[10,14] He has also recognized the need for control of the channel and proposed a method for controlling the transmission probability of "ready" packets.

Other multi-access broadcast packet switching schemes have been proposed to take advantage of special system and traffic characteristics. A reservation scheme studied by Roberts[5] employs a slotted ALOHA subchannel for broadcasting block transfer reservation requests. Reservation-ALOHA[2] and carrier sense multi-access[8] are both interesting variants of the random access scheme. These systems seem to exhibit unstable behavior similar to that of slotted ALOHA and may be dynamically controlled by algorithms similar to those presented in this paper. Consider, for instance, the ALOHA System at the University of Hawaii which uses two 24 KBPS radio channels and which has been

* This research was supported in part by the Advanced Research Projects Agency of the Department of Defense under Contract No. DAHC-15-73-C-0368.

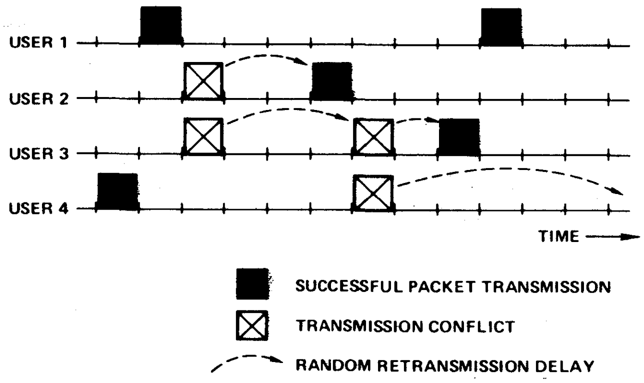


Figure 1—Slotted ALOHA random access

estimated to be able to support up to 500 interactive users.[1,3] We feel that this figure is unrealistic for an uncontrolled system, but may be achieved given some appropriate dynamic channel control.

THE RANDOM ACCESS CHANNEL MODEL

Consider a radio communication system such as a packet switching satellite system[3–7] or the ALOHA System.[1] In each case, there is a *broadcast* channel for point-to-multipoint communication and a *multi-access* channel shared by a large number of users. Since the broadcast channel is used by a single transmitter, no transmission conflict will arise. All nodes covered by the radio broadcast can receive on the same single frequency, picking out packet transmissions addressed to themselves and discarding packets addressed to others. The problem we are faced with is how to resolve conflicts which arise when "simultaneous" demands are placed upon the multi-access channel. If two or more packet transmissions overlap in time at the multi-accessed radio receiver (of the satellite transponder or the central computer), it is assumed that none is received correctly. This event will be referred to as a *channel collision*. The channel may be *slotted* by requiring all channel users to synchronize the leading edges of their packet transmissions at the multi-accessed radio receiver.[3–7] The duration of a channel time slot is set equal to a packet transmission time. In the slotted ALOHA random access scheme, all users transmit newly generated packets into channel time slots independently. In the event of a channel collision, each collided packet is retransmitted independently after a *retransmission delay* of RD slots. The above scheme is illustrated in Figure 1 for the case of a channel random-accessed by four users. (In a ground radio system, RD corresponds to the positive acknowledgment time-out interval.)

Consider a satellite multi-access broadcast system. Let R be the number of time slots in a round-trip satellite channel propagation time which is assumed to be the same for all earth stations. Thus, R time slots after transmitting a packet, a user will either hear that he was successful or know that he had a channel collision. (We have ignored the possibility of random noise errors assuming that the channel has a low error rate.) The retransmission delay RD for a collided packet must be greater than R. Randomization of RD is necessary to minimize the probability of repeated channel collisions for the same packets. Randomization schemes which have been considered include: (1) the uniform retransmission randomization scheme[4] in which the probability distribution of RD is given by

$$\text{Prob}\,[\text{RD}=\text{i}]=\begin{cases}0 & \text{i}\leq\text{R}\\ 1/\text{K} & \text{R}+1\leq\text{i}\leq\text{R}+\text{K}\\ 0 & \text{i}>\text{R}+\text{K}\end{cases}\qquad(1)$$

and (2) the geometric retransmission randomization scheme[6,7,10–14] in which the probability distribution of RD is given by

$$\text{Prob}\,[\text{RD}=\text{i}]=\begin{cases}0 & \text{i}\leq\text{R}\\ \text{p}(1-\text{p})^{\text{i}-\text{R}-1} & \text{i}>\text{R}\end{cases}\qquad(2)$$

The uniform retransmission randomization scheme is adopted in Reference 4. In that reference, R is taken to be 12 and each time slot is 22.5 milliseconds long, giving 44.4 slots/second. These figures are computed from the assumptions of a 50 KBPS satellite voice channel, 1125 bits/packet and a roundtrip channel propagation time of 0.27 second for all channel users. These same numerical constants are adopted in this paper. However, to study the problems of stability and dynamic channel control, it is necessary to consider a simplified Markovian model in which R=0 and the geometric retransmission randomization scheme is assumed, such that RD has a memoryless geometric distribution.[7,10–14] Simulation results have shown that the slotted ALOHA channel performance (in terms of average throughput and delay) is dependent primarily upon the average retransmission delay $\overline{\text{RD}}$ and quite insensitive to the exact probability distributions considered.[7] In order to use the analytic results of the Markovian model to predict the throughput-delay performance of a real slotted ALOHA channel with nonzero R, it is necessary to use a value of p in the Markovian model which matches the value of $\overline{\text{RD}}$. For example, to approximate the slotted ALOHA channel with uniform retransmission randomization and for which $\overline{\text{RD}}=\text{R}+(\text{K}+1)/2$, we must let

$$\text{p}=\frac{1}{\text{R}+(\text{K}+1)/2}\qquad(3)$$

such that $\overline{\text{RD}}$ is the same in both cases. Numerical results in this paper will always be expressed in terms of K (rather than p) through use of Equation (3).

Let us now introduce the Markovian model,[7,11–13] in which we consider a slotted ALOHA channel with a user population consisting of M users. Each such user can be in one of two states: *blocked* or *thinking*.[10,14] In the thinking state, a user generates (and transmits) a new packet in a time slot with probability σ. A packet which had a channel collision and is waiting for retransmission is said to be *backlogged*. The retransmission delay RD of each backlogged packet is assumed to be geometrically distributed, i.e., each backlogged packet retransmits in the current time slot with probability p. Assuming bursty users, we must have $\text{p}\gg\sigma$. From the time a user generates a packet until that packet is successfully received, the user is *blocked* in the sense that he cannot generate (or accept from his input source) a new packet for transmission.

Let N^t be a random variable (called the *channel backlog*) representing the total number of backlogged packets at time t. The "channel input" rate at time t is $S^t = (M - N^t)\sigma$. We shall assume M and σ to be time-invariant unless stated otherwise. In this case, N^t is a Markov process (chain) with stationary transition probabilities and serves as the state description for the system. The discrete *state space* consists of the set of integers $\{0, 1, 2, \ldots, M\}$.

CHANNEL STABILITY

In this section, we give a brief description of the stability behavior of an uncontrolled slotted ALOHA system studied earlier.[7,10–12,14] Consider the trajectory of (N^t, S^t) in the two-dimensional (n, S) plane. Assuming that M and σ are constant, (N^t, S^t) is constrained to lie on the straight line $S = (M - n)\sigma$ called the *channel load line.* Corresponding to a fixed value* of K, there is an *equilibrium contour* in the (n, S) plane defined as the locus of points for which the channel input rate S is exactly equal to the expected channel throughput (defined to be the probability of a successful packet transmission) $S_{\text{out}}(n, S)$ in a time slot. A family of such contours is illustrated in Figure 2. Let us focus upon an equilibrium contour corresponding to $K = K_o$ in Figure 3. In the shaded region enclosed by the equilibrium contour, $S_{\text{out}}(n, S)$ is greater than S; elsewhere, S exceeds $S_{\text{out}}(n, S)$. Arrows on the channel load lines point in the direction of "drift" of the channel backlog size N^t. Three channel load lines are also shown in Figure 3 corresponding to channel user population sizes M, M' and M'', and an average user think time of $1/\sigma$ slots.

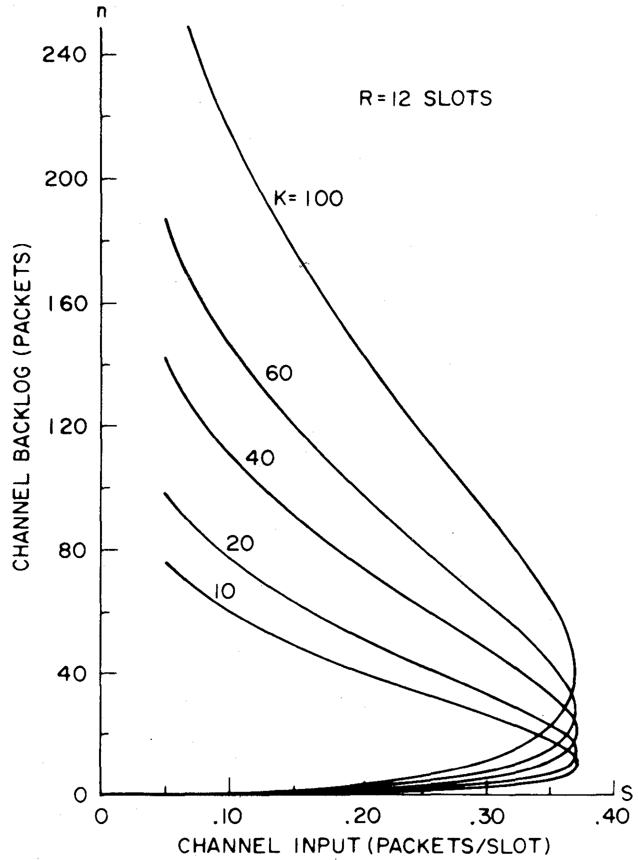
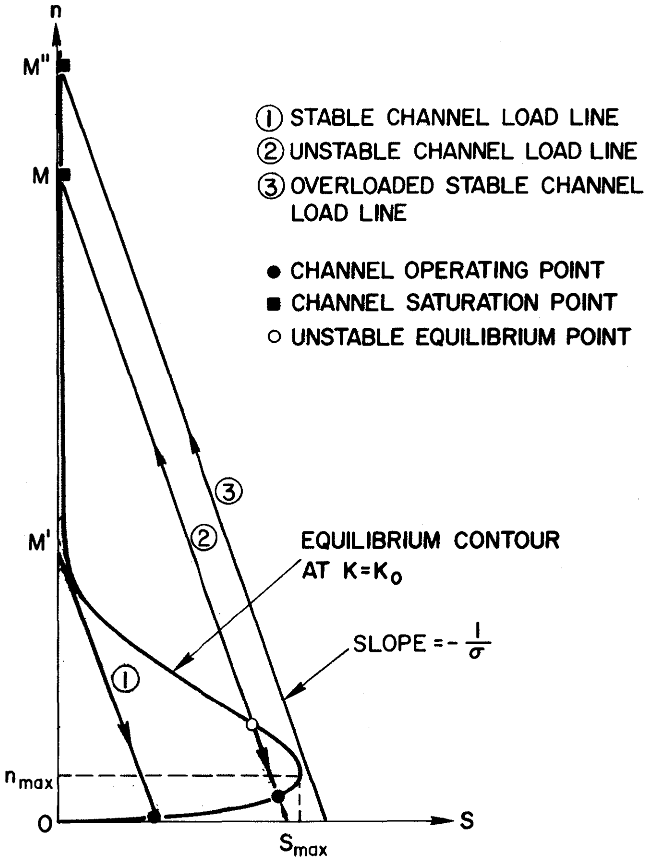


Figure 2—Equilibrium contours

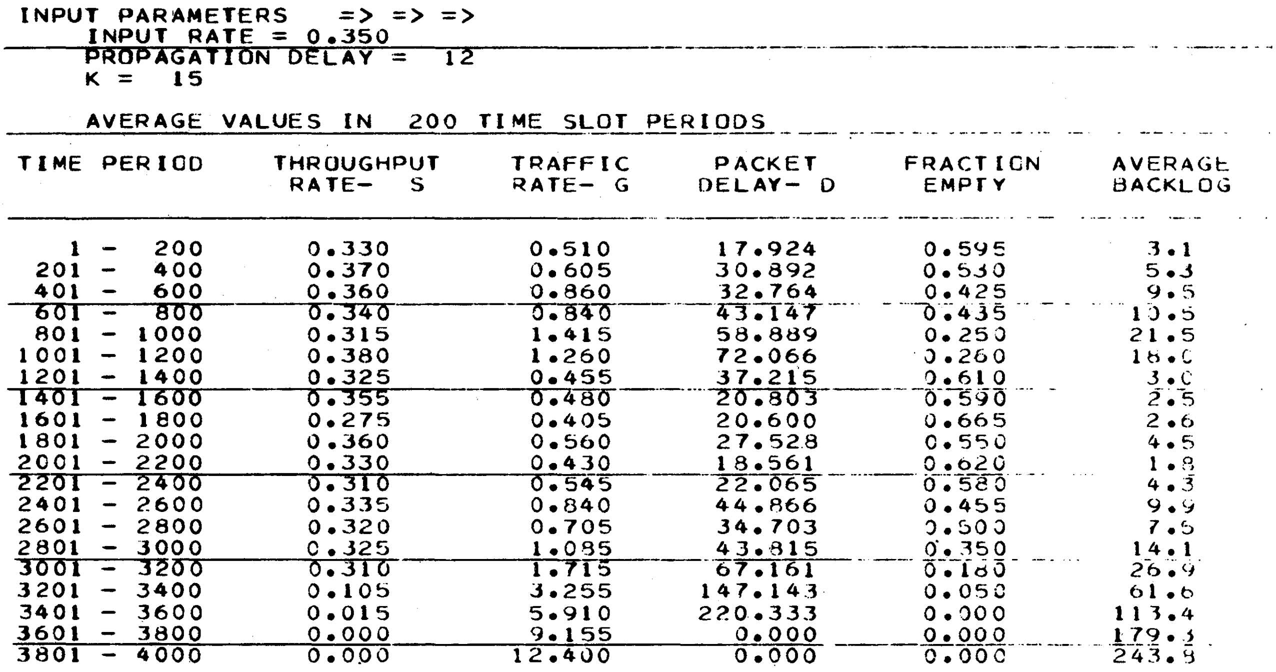


n
M''
M
M'
① STABLE CHANNEL LOAD LINE
② UNSTABLE CHANNEL LOAD LINE
③ OVERLOADED STABLE CHANNEL LOAD LINE
● CHANNEL OPERATING POINT
■ CHANNEL SATURATION POINT
○ UNSTABLE EQUILIBRIUM POINT
③
②
①
EQUILIBRIUM CONTOUR AT K=K0
SLOPE = - 1/σ
n max
0
S
S max

Figure 3—Stable and unstable channels

A channel load line may intersect the equilibrium contour one or more times, and we refer to these as equilibrium points which we denote by (n_e, S_e). An equilibrium point on a load line is said to be a *stable equilibrium point* if it acts as a "sink" with respect to the drift of N^t; an equilibrium point is said to be an *unstable equilibrium point* if it acts as "source." A stable equilibrium point is said to be the *channel operating point* if $n_e \leq n_{\max}$ as shown in Figure 3; it is said to be the *channel saturation point* if $n_e > n_{\max}$. (We shall use (n_o, S_o) instead of (n_e, S_e) to distinguish the channel operating point from other equilibrium points.) A channel load line is defined to be *stable* if it has exactly one stable equilibrium point; otherwise it is defined to be *unstable.* Thus, the load lines 1 and 3 in Figure 3 are stable by definition; the load line 2 is unstable.

* Or equivalently a fixed value of p under Equation (3).



```
INPUT PARAMETERS    => => =>
     INPUT RATE = 0.350
     PROPAGATION DELAY =  12
     K =  15

     AVERAGE VALUES IN  200 TIME SLOT PERIODS

TIME PERIOD        THROUGHPUT       TRAFFIC        PACKET        FRACTION       AVERAGE
                    RATE-  S        RATE- G      DELAY- D         EMPTY        BACKLOG

   1 -  200         0.330           0.510         17.924          0.595           3.1
 201 -  400         0.370           0.605         30.892          0.530           5.3
 401 -  600         0.360           0.860         32.764          0.425           9.5
 601 -  800         0.340           0.840         43.147          0.435          10.5
 801 - 1000         0.315           1.415         58.889          0.250          21.5
1001 - 1200         0.380           1.260         72.066          0.260          18.0
1201 - 1400         0.325           0.455         37.215          0.610           3.0
1401 - 1600         0.355           0.480         20.803          0.590           2.5
1601 - 1800         0.275           0.405         20.600          0.665           2.6
1801 - 2000         0.360           0.560         27.528          0.550           4.5
2001 - 2200         0.330           0.430         18.561          0.620           1.8
2201 - 2400         0.310           0.545         22.065          0.580           4.3
2401 - 2600         0.335           0.840         44.866          0.455           9.9
2601 - 2800         0.320           0.705         34.703          0.500           7.5
2801 - 3000         0.325           1.085         43.815          0.350          14.1
3001 - 3200         0.310           1.715         67.161          0.180          26.9
3201 - 3400         0.105           3.255        147.143          0.050          61.6
3401 - 3600         0.015           5.910        220.333          0.000         113.4
3601 - 3800         0.000           9.155          0.000          0.000         179.3
3801 - 4000         0.000          12.400          0.000          0.000         243.8
```

Figure 4—An unstable channel drifting into saturation

If M is finite, a stationary probability distribution always exists for N^t. In a stable channel, the equilibrium point (n_e, S_e) gives (approximately) the steady-state throughput-delay performance of the channel over an infinite time horizon. On the other hand, an unstable channel exhibits "bistable" behavior; the throughput-delay performance given by the channel operating point is achievable only for a finite time period before the channel drifts toward the channel saturation point. When this happens, the channel performance degrades rapidly as the channel throughput rate decreases and the average packet delay increases. In this state, the communication channel can be regarded as having failed. (In a practical system, external control should be applied at this point to restore proper channel operation.) In Figure 4, we have shown a simulation of the above behavior. In this example, M is assumed to be so large that the channel input is Poisson distributed at a constant rate $S=0.35$.

The channel load line labelled 3 in Figure 3 has a channel saturation point as its only stable equilibrium point. It is overloaded in the sense that M'' is too big for the given σ and K. From now on, a stable channel load line will always refer to 1 instead of 3.

Given a channel load line, suppose K_{opt} is the optimum K which minimizes n_o and maximizes S_o at the channel operating point. For this value of K, the channel may be unstable in which case the optimum channel performance given by (n_o, S_o) is achievable only for a finite time period. In References 7, 11 and 12, the average "up" time of an unstable channel has been quantified as a stability measure of the channel. To render the channel stable, two obvious solutions are available: (1) use a larger value for K (see Figure 2), and (2) reduce the user population size M. The first solution gives rise to a smaller S_o and a larger n_o; the corresponding average packet delay may then be too large to be acceptable. In the second solution, a small M implies that $S_o \ll S_{\max}$ (see Figure 3) since $\sigma \ll 1$ under the assumption of bursty users. This results in a waste of channel capacity.

The third solution is the use of dynamic channel control which constitutes the subject matter of the balance of this paper.

THE DYNAMIC CHANNEL CONTROL MODEL

To prevent the disastrous consequences of channel saturation, various dynamic control measures may be taken. In this section, we describe the dynamic channel control model studied in References 7 and 13, and outline some of the results obtained there under the assumption of *perfect channel state information*, i.e., each channel user knows the exact value of the channel backlog N^t at time t. In the next section, we shall consider practical control schemes which estimate the channel state and apply the theoretical optimal control policies using this estimate.

Consider the finite-state Markovian decision model obtained by injecting the following two classes of control actions into our earlier model for N^t:

(i) each packet arrival is accepted for transmission with probability β and rejected with probability $1-\beta$ where $0 \leq \beta \leq 1$ and $\beta \in \{\beta_1, \beta_2, \ldots, \beta_m\} \triangleq \mathcal{A}_1$;

(ii) each backlogged packet is retransmitted with probability γ where $0<\gamma<1$ and $\gamma \in \{\gamma_1, \gamma_2, \ldots, \gamma_k\} \triangleq \mathcal{A}_2$.

$\mathcal{A} \triangleq \mathcal{A}_1 \times \mathcal{A}_2$ is said to be the control action space. Three special cases have been studied extensively in References 7 and 13, namely,

(1) The Input Control Procedure (ICP) with $\mathcal{A}=\{0,1\} \times \{p_o\}$,

(2) The Retransmission Control Procedure (RCP) with $\mathcal{A}=\{1\} \times \{p_o, p_c\}$, and

(3) The Input-Retransmission Control Procedure (IRCP) with $\mathcal{A}=\{0,1\} \times \{p_o, p_c\}$.

In these control procedures, p_o corresponds to some K_o which optimizes the channel operating point of the given channel load line; p_c corresponds to some K_c which is sufficiently large to render the given channel load line stable.

A *control policy* f is defined to be any rule for choosing control actions in $\mathcal{A}$. The action α^t, at time t given by the policy f, specifies both the state transition probabilities and some predefined expected state transition cost for the tth time slot. Thus f determines both the evolution in time of N^t and the sequence of costs it incurs. Given a cost structure (denoted by δ), the *cost rate* $g_\delta(f)$ of N^t under a control policy f is defined to be the steady-state average cost per unit time incurred by N^t.

An important subclass of all policies is the class of stationary policies. A *stationary policy* is defined to be one which chooses an action at time t depending only upon the state of the process at that time. From well-known results in Markov decision theory, we know that (1) if f is a stationary policy, $g_\delta(f)$ is independent of the initial state of the process N^t, and (2) a stationary policy f^* exists, which minimizes $g_\delta(f)$ over the class of all policies. Thus in our search for an optimal control strategy, we can limit our attention to the class of stationary policies only.

As the process N^t evolves from one time slot to the next, various expected state transition costs may be incurred, such as

(1) the expected channel throughput in the tth time slot,

(2) the (delay) cost of holding backlogged packets, and

(3) the expected (delay) cost of rejecting packet arrivals.

Type 1 costs take on negative values since we want to maximize the channel throughput rate. Type 2 costs are chosen such that each backlogged packet incurs 1 unit of delay per time slot. In the references, the expected cost in units of delay per packet arrival rejected (type 3 costs) is assumed to be equal to an average user think time. This assumption is needed for our Markovian model formulation and may be justified in a terminal access communications environment as follows. A person sitting at a terminal generates a new packet with an average think time of $1/\sigma$ whenever his previous packet has been successfully transmitted. If, at the time of a packet arrival, the channel is in the reject state, this packet is lost in the sense that it is not transmitted over the channel immediately. In a practical situation, the user may be informed of the event and must enter some command character to "resend" the packet. Hence, the cost in terms of delay is probably in the order of an average think time $(=1/\sigma)$.

Let $g_s(f)$ denote the cost rate of N^t given by policy f and type 1 costs, and $g_d(f)$ denote the cost rate of N^t given by policy f and types 2 and 3 costs. The channel performance measures, namely, the *steady-state channel throughput rate* S_{out} and the *expected packet delay* D can then be calculated in terms of $g_s(f)$ and $g_d(f)$.

In the references, it is shown that for the given model an optimal stationary control policy maximizes S_{out} and minimizes D simultaneously. An efficient computational algorithm (POLITE) based upon Howard's policy-iteration method[15] is given for calculating the optimal policy. Given a channel load line and a dynamic control procedure ($\mathcal{A}$), this algorithm usually arrives at the optimal control policy and the optimum values of S_{out} and D in very few iterations. Furthermore, numerical results indicate that each optimal

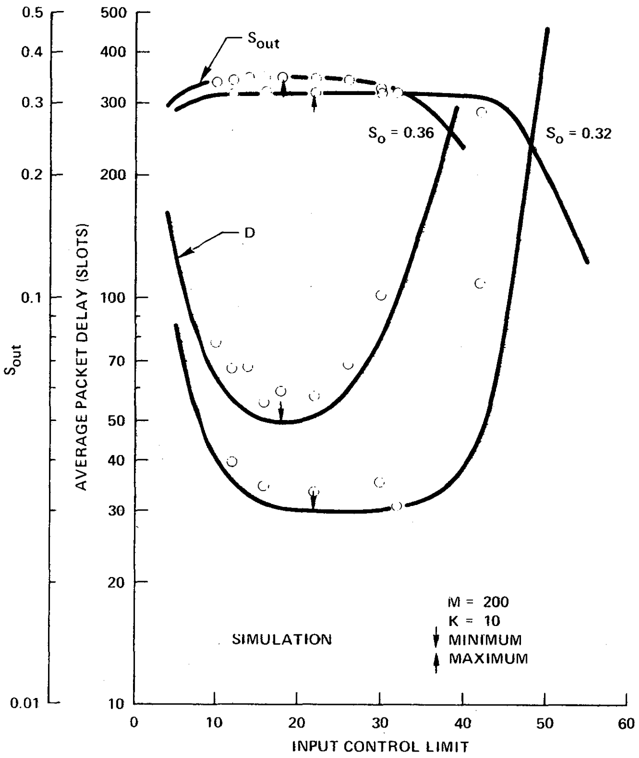


Figure 5—Channel performance versus ICP control limit for $M=200$

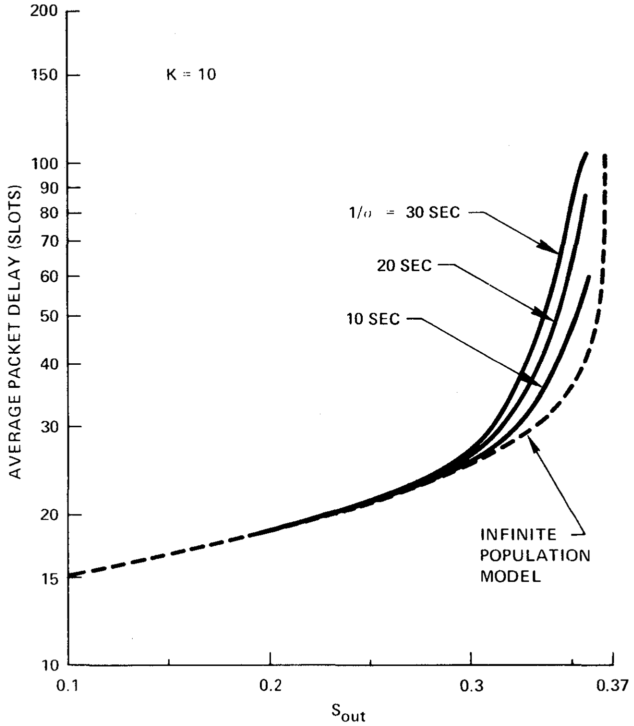


Figure 6—ICP optimum throughput-delay tradeoffs at fixed σ

control policy f for the control procedures ICP and RCP has the following structure:

$$f(i)=\begin{cases} a_o & 0\leq i\leq \hat{n} \\ a_c & \hat{n}<i\leq M \end{cases} \qquad (4)$$

where a_o corresponds to "accept" in ICP and "p_o" in RCP; a_c corresponds to "reject" in ICP and "p_c" in RCP. On the other hand, an optimal control policy f for IRCP has the following structure:

$$f(i)=\begin{cases} (\text{accept}, p_o) & 0\leq i\leq \hat{n}_1 \\ (\text{accept}, p_c) & \hat{n}_1<i\leq \hat{n}_2 \\ (\text{reject}, p_c) & \hat{n}_2<i\leq M \end{cases} \qquad (5)$$

We shall refer to $\hat{n}$, $\hat{n}_1$ and $\hat{n}_2$ as control limits and the control policies in Equations (4) and (5) as *control limit policies.*

In Figure 5, we have shown the performance measures, S_{out} and D, for two channel load lines specified by $M=200$ and the channel operating point $(n_o, S_o)=(4,0.32)$ and $(7,0.36)$, over a range of ICP control limit policies. Observe that the same control limit minimizes D and maximizes S_{out} at the same time as predicted by the theory. Note the amazing flatness of S_{out} and D near the optimum point for the channel load line with $S_o=0.32$. The consequence is that even if a nonoptimal control policy is used (due, for example, to not knowing the exact current backlog size such as in most practical systems), it is still possible to achieve a throughput-delay performance close to the optimum.

In Figure 5, we have also shown simulation results for throughput and delay. In these simulations, channel control policies are applied assuming that the exact channel backlog size N^t is known to all channel users. However, contrary to the Markovian model, each collided packet is assumed to suffer the more realistic fixed delay R and its retransmission is randomized uniformly over the next K slots. The excellent agreement between the simulation and analytic results presented here demonstrates the accuracy of the approximation.

In Figure 6, we show optimum throughput-delay tradeoffs at fixed values of σ for ICP. ($1/\sigma$ is the average think time of a channel user.) In this case, increasing S_{out} corresponds to increasing M, that is, admitting more channel users. We see that the channel performance improves as the packet generation probability σ increases, since this implies that for the same S_{out}, the number of channel users M is smaller. In the latter case, the channel is "less unstable."[7,11,12]

PRACTICAL CONTROL SCHEMES

In a practical system, the channel users often have no means of communication among themselves other than the multi-access broadcast channel itself. Each channel user must individually estimate the channel state by observing the outcome in each channel slot. Moreover, whatever channel state information available to the channel users is at least one round-trip propagation delay (R) old and may introduce additional errors in the users' estimates if R is large (such as in a satellite channel). Thus, the control action applied based upon an estimate of the channel state may not necessarily be the optimal one at that time, which then will lead to some degradation in channel performance.

Below we first give a heuristic scheme for estimating the channel state assuming that the channel history (i.e., empty slots, successful transmissions or collisions) is available to all channel users. The optimal ICP, RCP and IRCP control policies will be applied based upon the above estimate. A heuristic control procedure is next proposed which circumvents the state estimation problem. These control procedures are then examined through simulation and compared with the optimum throughput-delay results in the previous section. The ability of these control procedures to handle time-varying inputs (with pulses) is also examined.

Channel control-estimation (CONTEST) algorithms

The *channel traffic* in a time slot is defined to be the number of packet transmissions (both new and previously collided packets) by all users in that time slot. Our heuristic procedure for estimating the channel state is based upon the observation that the channel traffic in a time slot is approximately Poisson distributed. (See Chapter 4 and Appendix A of Reference 7.) Below we present algorithms which implement channel control procedures studied in the previous sections using estimates of the channel state. These channel CONTrol-ESTimation algorithms will be referred to as CONTEST algorithms.

Here we give a procedure for implementing RCP. Suppose $\hat{n}$ is the RCP control limit such that the channel users switch their retransmission K value from K_o to K_c when the channel backlog size exceeds $\hat{n}$ and from K_c to K_o as soon as the channel backlog size drops below $\hat{n}$. We define

$$\hat{G}_o=\hat{n}p_o+(M-\hat{n})\sigma \tag{6}$$

and

$$\hat{G}_c=\hat{n}p_c+(M-\hat{n})\sigma \tag{7}$$

$\hat{G}_o$ and $\hat{G}_c$ represent the average channel traffic rates given that the channel backlog size is $\hat{n}$ packets with K equal to K_o and K_c respectively. Assuming that the channel traffic is approximately Poisson distributed, we define the following critical values (corresponding to the probability of zero channel traffic in a time slot),

$$\hat{f}_o=e^{-\hat{G}_o} \tag{8}$$

and

$$\hat{f}_c=e^{-\hat{G}_c} \tag{9}$$

Since $K_c>K_o$ we must have

$$\hat{f}_o<\hat{f}_c$$

Suppose each channel user keeps track of the channel history (one round-trip propagation delay ago) within a window frame of W slots. Let $\bar{f}^t$ be the measured fraction of empty slots in the W slots within the history window for the tth time slot. $\bar{f}^t$ will closely approximate the probability of zero channel traffic in the tth time slot provided that the channel traffic probability distribution does not change appreciably in $(W+R)$ time slots, that $W\gg 1$ and the Poisson traffic assumption holds. We give the following algorithm to be adopted by each channel user. d^t denotes the control decision at time t.

Algorithm 1 (*RCP-CONTEST*)—This algorithm generates the decision $d^t = K_o$, K_c at each time point based upon the channel state estimate $\bar{f}^t$ and the RCP control limit $\hat{n}$. Start at step (1) or step (4).

```
(1) $t \leftarrow t+1$
    $d^t = K_o$
(2) If $\bar{f}^t<\hat{f}_o$, go to (4)
(3) Go to (1)
(4) $t\leftarrow t+1$
    $d^t=K_c$
(5) If $\bar{f}^t>\hat{f}_c$, go to (1)
(6) Go to (4)
```

Next we consider a similar implementation for ICP. We define

$$\hat{G}_a=\hat{n}p+(M-\hat{n})\sigma \tag{10}$$

$$\hat{G}_r=\hat{n}p \tag{11}$$

$$\hat{f}_a=e^{-\hat{G}_a} \tag{12}$$

and

$$\hat{f}_r=e^{-\hat{G}_r} \tag{13}$$

Since $\hat{G}_a>\hat{G}_r$, we must have

$$\hat{f}_a<\hat{f}_r$$

TABLE I—Throughput-delay Results of a Controlled Channel ($M=200$, $S_o=0.32$)

CONTROL SCHEME	S_{out}	D
ICP (POLITE)	0.31778	29.857
RCP (POLITE)	0.31817	29.085
IRCP(POLITE)	0.31817	29.085
ICP (Simulation)	0.315	33.427
RCP (Simulation)	0.318	28.824
ICP-CONTEST W = 20	0.314	40.893
ICP-CONTEST W = 40	0.315	30.514
ICP-CONTEST W = 60	0.317	32.355
ICP-CONTEST W = 80	0.318	35.809
RCP-CONTEST W = 20	0.315	33.052
RCP-CONTEST W = 40	0.322	33.335
RCP-CONTEST W = 60	0.319	32.138
RCP-CONTEST W = 80	0.317	32.501
Heuristic RCP $K_1 = 10$ $K_m = 60$ $m\geq 2$	0.316 0.315	33.720 34.554
Heuristic RCP $K_1 = 10$ $K_2 = 60$ $K_m = 120$ $m\geq 3$	0.310 0.316	35.425 34.635

Algorithm 2 (*ICP-CONTEST*)—This algorithm generates the decision d^t=accept, reject at time t, based upon the channel state estimate $\bar{f}^t$ and ICP control limit $\hat{n}$. Start at step (1) or step (4).

```
(1) $t\leftarrow t+1$
    $d^t$=accept
(2) If $\bar{f}^t<\hat{f}_a$ go to (4)
(3) Go to (1)
(4) $t\leftarrow t+1$
    $d^t$=reject
(5) If $\bar{f}^t>\hat{f}_r$ go to (1)
(6) Go to (4)
```

Finally, to implement IRCP, we assume that the control policy is of the form given in Equation (5) such that it is uniquely specified by the control limits $\hat{n}_1$ and $\hat{n}_2$. We define $\hat{f}_o$ and $\hat{f}_c$ by using $\hat{n}_1$ in Equations (6)-(9), $\hat{f}_{ac}$ and $\hat{f}_{rc}$ by using $\hat{n}_2$ and p_c in Equations (10)-(13) and $\hat{f}_{ao}$ by using $\hat{n}_2$ and p_o in Equations (10) and (12). Since $p_o>p_c>\sigma$ and $\hat{n}_2>\hat{n}_1$, we have $\hat{f}_{ao}<\hat{f}_o$ and $\hat{f}_{ac}<\hat{f}_c$.

Algorithm 3 (*IRCP-CONTEST*)—This algorithm generates the decision d^t= (accept, K_o), (accept, K_c), (reject, K_c)

TABLE II—Throughput-delay Results of a Controlled Channel ($M=400$, $S_o=0.32$)

CONTROL SCHEME		S_{out}	D
ICP (POLITE)		0.31807	33.096
RCP (POLITE)		0.31844	31.608
IRCP (POLITE)		0.31844	31.608
ICP (Simulation)		0.315	31.427
RCP (Simulation)		0.317	31.023
ICP-CONTEST	W = 20	0.315	43.262
ICP-CONTEST	W = 40	0.314	34.723
ICP-CONTEST	W = 60	0.312	53.240
ICP-CONTEST	W = 80	0.316	39.112
RCP-CONTEST	W = 20	0.313	41.087
RCP-CONTEST	W = 40	0.319	43.379
RCP-CONTEST	W = 60	0.318	38.821
RCP-CONTEST	W = 80	0.317	40.068
RCP-CONTEST	W = 100	0.314	35.689
RCP-CONTEST	W = 120	0.319	47.149
Heuristic RCP	K_1 = 10	0.316	45.150
	K_m = 150 $m\geq 2$	0.316	44.750
Heuristic RCP	K_1 = 10		
	K_2 = 100	0.312	42.040
	K_m = 200 $m\geq 3$	0.311	43.136

at time t based upon the channel state estimate $\hat{f}^t$ and IRCP control policy ($\hat{n}_1$, $\hat{n}_2$). Start at step (1), (4), or (7).

```
(1) $t \leftarrow t+1$
    $d^t=$ (accept, $K_o$)
(2) If $\hat{f}^t<\hat{f}_{ao}$ go to (7)
    otherwise, if $\hat{f}^t<\hat{f}_o$ go to (4)
(3) go to (1)
(4) $t \leftarrow t+1$
    $d^t=$ (accept, $K_c$)
(5) If $\hat{f}^t>\hat{f}_c$ go to (1)
    otherwise, if $\hat{f}^t<\hat{f}_{ac}$ go to (7)
(6) go to (4)
(7) $t \leftarrow t+1$
    $d^t=$ (reject, $K_c$)
(8) If $\hat{f}^t>\hat{f}_{rc}$ go to (4)
(9) go to (7)
```

The size W of the channel history window kept by each channel user is very important for successful channel state estimation. If W is too large, we may lose information on the dynamic behavior of the channel such that the necessary actions are taken too late. If W is too small, we may get large errors in approximating the probability of zero channel traffic by the fraction of empty slots in the history window. A good initial estimate is that W should be bigger than R and of the same order of magnitude. Below we compare simulation results on channel performance for different values of W.

Another retransmission control procedure

In this section we describe a simple heuristic control procedure which has the property that when the channel traffic increases the retransmission delays of backlogged packets will also increase. Hence, it will be referred to as the heuristic retransmission control procedure (*Heuristic RCP*). The advantage of such a control procedure is that it is simple and can be implemented easily without any need for monitoring the channel history and estimating the channel state.

Algorithm 4 (*Heuristic RCP*)—For a backlogged packet with m previous channel collisions, the uniform retransmission randomization* interval is taken to be $K=K_m$ where K_m is a monotone nondecreasing function in m.

When the channel traffic increases, the probability of channel collision increases. As a result, the "effective" value of K increases. If K_m is a steep enough function of m, we see that channel saturation will be prevented. An effective value of K can be defined only with respect to a specific performance measure (e.g., average packet delay). To illustrate the effect of the function K_m, we derive below the *average* value of K as a function of q (the probability of successful transmission) for two cases. Let

$$\begin{aligned} r_i &= \text{Prob [a packet retransmits } i \text{ times before success]} \\ &= (1-q)^i q \qquad i\geq 1 \end{aligned}$$

Case 1 $\quad K_m=K_2$ for $m\geq 2$ and $K_2>K_1$

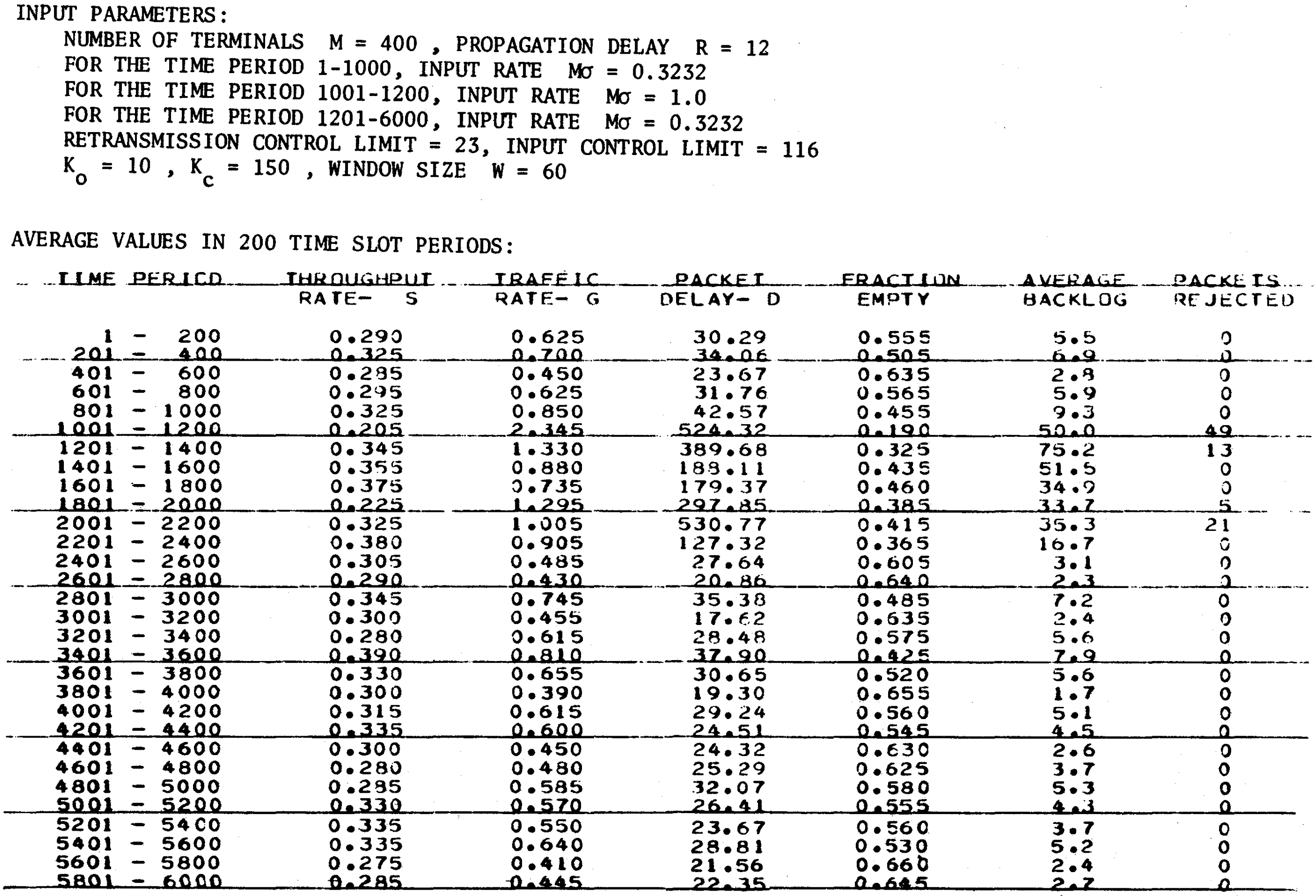
$$\begin{aligned} \bar{K} &= \text{average value of } K \\ &= \frac{1}{1-q}\sum_{i=1}^{\infty} r_i \sum_{m=1}^{i} \frac{K_m}{i} \\ &= \frac{1}{1-q}\sum_{i=1}^{\infty} (1-q)^i q\left(\frac{K_1}{i}+\frac{i-1}{i}K_2\right) \\ &= K_2+\frac{q\ln q}{1-q}(K_2-K_1) \qquad (14) \end{aligned}$$

which is equal to K_1 at $q=1$ and increases to K_2 as q decreases to zero; ln is the natural logarithm function.

Case 2 $\quad K_m=mK \qquad m\geq 1$

$$\begin{aligned} \bar{K} &= \frac{1}{1-q}\sum_{i=1}^{\infty} r_i \sum_{m=1}^{i} \frac{K_m}{i} \\ &= \frac{K}{1-q}\sum_{i=1}^{\infty} \frac{(1-q)^i q}{i} \sum_{m=1}^{i} m \\ &= \frac{K}{2}\left(1+\frac{1}{q}\right) \qquad (15) \end{aligned}$$

* Note that the same control scheme can be extended to geometric retransmission randomization by letting $p = p_m$ where p_m is a monotone nonincreasing function in m.



```
INPUT PARAMETERS:
    NUMBER OF TERMINALS  M = 400 , PROPAGATION DELAY  R = 12
    FOR THE TIME PERIOD 1-1000, INPUT RATE  Mσ = 0.3232
    FOR THE TIME PERIOD 1001-1200, INPUT RATE  Mσ = 1.0
    FOR THE TIME PERIOD 1201-6000, INPUT RATE  Mσ = 0.3232
    RETRANSMISSION CONTROL LIMIT = 23, INPUT CONTROL LIMIT = 116
    Ko = 10 , Kc = 150 , WINDOW SIZE  W = 60

AVERAGE VALUES IN 200 TIME SLOT PERIODS:
  TIME PERICD    THROUGHPUT    TRAFFIC     PACKET      FRACTION   AVERAGE    PACKETS
                 RATE-  S      RATE- G     DELAY- D    EMPTY      BACKLOG    REJECTED

     1 -  200      0.290        0.625        30.29       0.555       5.5         0
   201 -  400      0.325        0.700        34.06       0.505       6.9         0
   401 -  600      0.285        0.450        23.67       0.635       2.8         0
   601 -  800      0.295        0.625        31.76       0.565       5.9         0
   801 - 1000      0.325        0.850        42.57       0.455       9.3         0
  1001 - 1200      0.205        2.345       524.32       0.190      50.0        49
  1201 - 1400      0.345        1.330       389.68       0.325      75.2        13
  1401 - 1600      0.355        0.880       188.11       0.435      51.5         0
  1601 - 1800      0.375        0.735       179.37       0.460      34.9         0
  1801 - 2000      0.225        1.295       297.85       0.385      33.7         5
  2001 - 2200      0.325        1.005       530.77       0.415      35.3        21
  2201 - 2400      0.380        0.905       127.32       0.365      16.7         0
  2401 - 2600      0.305        0.485        27.64       0.605       3.1         0
  2601 - 2800      0.290        0.430        20.86       0.640       2.3         0
  2801 - 3000      0.345        0.745        35.38       0.485       7.2         0
  3001 - 3200      0.300        0.455        17.62       0.635       2.4         0
  3201 - 3400      0.280        0.615        28.48       0.575       5.6         0
  3401 - 3600      0.390        0.810        37.90       0.425       7.9         0
  3601 - 3800      0.330        0.655        30.65       0.520       5.6         0
  3801 - 4000      0.300        0.390        19.30       0.655       1.7         0
  4001 - 4200      0.315        0.615        29.24       0.560       5.1         0
  4201 - 4400      0.335        0.600        24.51       0.545       4.5         0
  4401 - 4600      0.300        0.450        24.32       0.630       2.6         0
  4601 - 4800      0.280        0.480        25.29       0.625       3.7         0
  4801 - 5000      0.285        0.585        32.07       0.580       5.3         0
  5001 - 5200      0.330        0.570        26.41       0.555       4.3         0
  5201 - 5400      0.335        0.550        23.67       0.560       3.7         0
  5401 - 5600      0.335        0.640        28.81       0.530       5.2         0
  5601 - 5800      0.275        0.410        21.56       0.660       2.4         0
  5801 - 6000      0.285        0.445        22.35       0.645       2.7         0
```

Figure 7—Simulation run for IRCP-CONTEST subject to a channel input pulse

which is equal to K at $q=1$ and increases to infinity as q decreases to zero.

The above results indicate that the average value of K behaves in the desired manner, namely, $\bar{K}$ increases as q decreases due to an increasing channel traffic. Below we examine the CONTEST algorithms and Heuristic RCP through simulations.

Simulation results

We summarize in Tables I-II, throughput-delay results for channel load lines specified by

(1) $M=200$, $(n_o, S_o)=(4, 0.32)$
(2) $M=400$, $(n_o, S_o)=(4, 0.32)$

In both cases, we assume $K_o=10$ and $K_c=60$. Included in these tables are (a) optimum POLITE results for ICP, RCP and IRCP, (b) simulation results for ICP and RCP using optimal control policies and under the assumption of perfect channel state information, (c) simulation results for the CONTEST algorithm using ICP and RCP optimal control policies, and (d) simulation results for Heuristic RCP. The duration of each simulation run was taken to be 30,000 time slots. IRCP was not tested by simulation since the optimal value of $\hat{n}_2$ is in all cases so large that within the simulation duration, the channel state N^t (almost surely) will not exceed it; the control procedure becomes effectively RCP specified by $\hat{n}_1$.

The ICP-CONTEST algorithm was tested with channel history window sizes of 20, 40, 60, and 80 time slots. We see from Tables I and II that $W=40$ appears to give the best throughput-delay results. Note that for $R=12$ and $K=10$, $W=40$ is approximately twice $R+K$.

The RCP-CONTEST algorithm was also tested with various values of W. In this case, K takes on two values, K_o and K_c. There is no clear-cut optimal W. It appears that $W=60$ is a good choice.

There is no significant degradation in channel performance (from the theoretical optimum) given by the CONTEST algorithms and Heuristic RCP. The CONTEST algorithms, however, seem to have an edge over Heuristic RCP. The excellent performance of the CONTEST algorithms can be attributed to the flatness of S_{out} and D near the optimum as a function of the control limit (see Figure 5). We found that this flatness property is less pronounced for channel load lines with large values of S_o or M, such as $S_o=0.36$ or

```
INPUT PARAMETERS:
    NUMBER OF TERMINALS  M = 400 , PROPAGATION DELAY  R = 12
    FOR THE TIME PERIOD 1-1000, INPUT RATE  Mσ = 0.3232
    FOR THE TIME PERIOD 1001-1200, INPUT RATE  Mσ = 1.0
    FOR THE TIME PERIOD 1201-6000, INPUT RATE  Mσ = 0.3232
    K  = 10      K  = 150 (m ≥ 2)
     1            m

AVERAGE VALUES IN 200 TIME SLOT PERIODS:

   TIME PERIOD      THROUGHPUT    TRAFFIC     PACKET      FRACTION    AVERAGE
                     RATE-  S    RATE- G    DELAY- D      EMPTY      BACKLOG

      1 -  200        0.285       0.395      19.877       0.665        2.1
    201 -  400        0.320       0.390      16.328       0.650        1.2
    401 -  600        0.255       0.425      22.824       0.660        2.8
    601 -  800        0.290       0.475      26.172       0.630        4.0
    801 - 1000        0.325       0.570      28.554       0.570        5.7
   1001 - 1200        0.230       2.395      34.109       0.120       68.8
   1201 - 1400        0.285       1.695     141.333       0.215      112.6
   1401 - 1600        0.310       1.500     273.177       0.230       91.8
   1601 - 1800        0.375       1.415     288.693       0.190       68.5
   1801 - 2000        0.280       1.110     224.661       0.375       53.1
   2001 - 2200        0.360       1.240     257.333       0.300       48.8
   2201 - 2400        0.355       0.925     193.986       0.395       31.3
   2401 - 2600        0.385       0.655     122.818       0.490       15.2
   2601 - 2800        0.320       0.565      68.094       0.565        8.8
   2801 - 3000        0.280       0.420      39.357       0.660        5.6
   3001 - 3200        0.295       0.495      31.678       0.615        6.3
   3201 - 3400        0.265       0.680      45.000       0.545       11.7
   3401 - 3600        0.350       0.750      37.057       0.485       13.3
   3601 - 3800        0.310       0.465      65.274       0.625        8.2
   3801 - 4000        0.275       0.520      33.618       0.610        7.7
   4001 - 4200        0.330       0.480      34.652       0.595        5.2
   4201 - 4400        0.325       0.615      29.585       0.540        7.5
   4401 - 4600        0.370       0.525      38.608       0.560        7.6
   4601 - 4800        0.260       0.705      44.250       0.550       15.9
   4801 - 5000        0.375       0.720      63.520       0.460       11.1
   5001 - 5200        0.350       0.635      41.729       0.520        9.0
   5201 - 5400        0.285       0.475      29.368       0.625        6.6
   5401 - 5600        0.315       0.510      36.460       0.595        4.9
   5601 - 5800        0.290       0.425      24.190       0.650        4.1
   5801 - 6000        0.305       0.490      28.738       0.610        4.7
```

Figure 8—Simulation run for heuristic RCP subject to a channel input pulse

$M=400$. This explains the more significant degradation in channel performance given by the CONTEST algorithms shown in Table II for $M=400$ than in Table I for $M=200$.

In Figure 4, it was shown that in an uncontrolled slotted ALOHA channel, a channel input rate of 0.35 packet/slot was enough to cripple the channel indefinitely. In Figures 7 and 8, we show by simulation that under severe pulse overload circumstances both the IRCP-CONTEST algorithm and Heuristic RCP prevented the channel from going into saturation. In these simulations, the normal channel load line was given by $M=400$ and $(n_o, S_o)=(4, 0.32)$ both before and after the pulse. During a period of 200 slots (namely, the time period 1000-1200 in the figures), the packet generation probability σ was increased such that $M\sigma=1.0$ packet/slot. Observe that both algorithms handled the sudden influx of new packets with ease. In both cases, the channel throughput, instead of vanishing to zero as in an uncontrolled channel, maintained at a high rate and within less than 3000 slots, the channel returned to almost normal operation.

Further discussions of results

In a real system, the channel input source will typically vary slowly with time; for example, the number of users fluctuates during the day. We must emphasize the fact that the control policies considered have been optimized to control statistical channel fluctuations under the assumption of a stationary channel input. Although we have shown that they can temporarily handle very high channel input rates, additional control mechanisms should be designed into the system to make sure that channel overload conditions do not prevail for any long period of time (e.g., by limiting the maximum number of users who can "sign-on" and become active channel users).

The control action space of IRCP includes both control action spaces of ICP and RCP as subsets. Thus IRCP must give a channel performance at least as good as ICP and RCP. Next, comparing IRCP-CONTEST and Heuristic RCP, we see that the latter is easier to implement. However, under a normal load (say $S_o \leq 0.32$), IRCP-CONTEST is superior to Heuristic RCP. This is because Heuristic RCP introduces longer delays to collided packets even when these packets are merely unlucky in light channel traffic. On the other hand, with IRCP, control actions are not exerted until the channel traffic exceeds certain "dangerous" levels.

CONCLUSIONS

Packet switched satellite and ground radio systems have been proposed as new alternatives for computer communications. A multi-access broadcast packet switching technique that has attracted considerable interest is the slotted ALOHA random access scheme. A slotted ALOHA channel multi-accessed by a large population of users has been shown to exhibit unstable behavior. Dynamic control schemes are necessary to prevent the occurrence of channel saturation in unstable channels. The dynamic channel control problem has been studied using a finite-state Markovian decision model in References 7 and 13 under the assumption of perfect channel state information.

In this paper we have studied dynamic channel control algorithms (CONTEST algorithms) which implement the theoretical control policies by using a heuristic scheme to estimate the instantaneous channel state. A heuristic retransmission control algorithm has also been studied which circumvents the state estimation problem. Simulation results indicate that these control algorithms are capable of achieving a channel throughput-delay performance close to the theoretical optimum, as well as capable of preventing channel saturation under temporary overload conditions.

The problem of unstable behavior is very real in random access systems (e.g., ALOHA, slotted ALOHA, reservation-ALOHA, carrier sense multi-access, etc.). To guarantee an acceptable level of channel performance for such systems, some form of dynamic channel control is a must. The probabilistic model and dynamic channel control schemes introduced herein for a slotted ALOHA channel can probably be extended to solve stability and dynamic control problems of other random access systems.

REFERENCES

1. Abramson, N., "THE ALOHA SYSTEM—Another Alternative for Computer Communications," Fall Joint Computer Conference, *AFIPS Conference Proceedings*, 1970, Vol. 37, pp. 281-285.
2. Crowther, W., R. Rettberg, D. Walden, S. Ornstein and F. Heart, "A System for Broadcast Communication: Reservation-ALOHA," *Proceedings of the Sixth Hawaii International Conference on System Sciences*, University of Hawaii, Honolulu, January, 1973.
3. Abramson, N., "Packet Switching with Satellites," National Computer Conference, New York, June 4-8, 1973, *AFIPS Conference Proceedings*, 1973, Vol. 42, pp. 695-702.
4. Kleinrock, L. and S. S. Lam, "Packet-Switching in a Slotted Satellite Channel," National Computer Conference, New York, June 4-8, 1973, *AFIPS Conference Proceedings*, 1973, Vol. 42, pp 703-710.
5. Roberts, L. G., "Dynamic Allocation of Satellite Capacity Through Packet Reservation," National Computer Conference, New York, June 4-8, 1973, *AFIPS Conference Proceedings*, 1973, Vol. 42, pp. 711-716.
6. Butterfield, S., R. Rettberg and D. Walden, "The Satellite IMP for the ARPA Network," Seventh Hawaii International Conference on System Sciences, University of Hawaii, Honolulu, January 8-10, 1974, *Proceedings of the Special Subconference on Computer Nets*, 1974.
7. Lam, S. S., *Packet Switching in a Multi-Access Broadcast Channel with Application to Satellite Communication in a Computer Network*, Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, March 1974 (available as Technical Report UCLA-ENG-7429, April 1974).
8. Kleinrock, L. and F. A. Tobagi, "Carrier Sense Multiple Access for Packet Switched Radio Channels," *Proc. of the International Conference on Communications*, Minneapolis, Minn., June 1974.
9. Jackson, P. E. and C. D. Stubbs, "A Study of Multi-access Computer Communications," Spring Joint Computer Conference, *AFIPS Conference Proceedings*, 1969, Vol. 34, pp. 491-504.
10. Metcalfe, R. M., "Steady-State Analysis of a Slotted and Controlled ALOHA System with Blocking," *Proceedings of the Sixth Hawaii International Conference on System Sciences*, University of Hawaii, Honolulu, January 1973.
11. Kleinrock, L. and S. S. Lam, "On Stability of Packet Switching in a Random Multi-Access Broadcast Channel," Seventh Hawaii International Conference on System Sciences, University of Hawaii, Honolulu, January 8-10, 1974, *Proceedings of the Special Subconference on Computer Nets*, 1974.
12. Kleinrock, L. and S. S. Lam, "Packet Switching in a Multi-Access Broadcast Channel: Performance Evaluation," *IEEE Transactions on Communications*, Vol. COM-23, April 1975.
13. Lam, S. S. and L. Kleinrock, "Packet Switching in a Multi-Access Broadcast Channel: Dynamic Control Procedures," to appear in *IEEE Transactions on Communications*.
14. Metcalfe, R. M., *Packet Communications*, Ph.D. Dissertation, Harvard University, 1973 (available as MIT Project MAC Technical Report TR-114, December 1973).
15. Howard, R., *Dynamic Probabilistic Systems Vol. 1: Markov Models and Vol. 2: Semi-Markov and Decision Processes*, Wiley, New York, 1971.

Operating system design considerations for the packet-switching environment*

by DAVID L. RETZ
Speech Communications Research Laboratory, Inc.
Santa Barbara, California

INTRODUCTION

One of the striking developments in computing and communication technology during the past decade is reflected in the evolution of packet-switching computer networks.[1,2,3] Packet-switching communication techniques allow dynamic allocation of a set of communication resources (circuits) so that they may be flexibly shared among a number of autonomous processors. Implementation of such packet-switching networks has required many design decisions, such as the choice of network topology, routing strategies, and the establishment of conventions, or protocols, for information interchange between network resources.

This paper is concerned with the design requirements of Host operating systems: those systems whose primary business is the management of computing resources rather than communication resources. Low-level communication tasks such as routing fall outside the realm of the Host responsibilities discussed here and are performed by means of a sub-network of small computers dedicated to the task of packet-switching. In the ARPANET these computers are called Interface Message Processors, or IMPS, and use packet-switching techniques to communicate via 50-kilobit common carrier circuits. Each IMP provides up to four high-speed synchronous serial ports to which Hosts connect using special-purpose Host-IMP interfaces.[4]

Packet-switching network environments place special requirements on the design of the connected Host operating systems. Attachment to the ARPANET, for example, has required a number of additions or modifications to existing operating systems. There are certain structural features which must be incorporated in system design in order to facilitate effective use of distributed computing resources. We begin by examining a few of these features.

IMPLICATIONS ON HOST SYSTEM ARCHITECTURE

Sharing of distributed resources is made possible by the cooperation of distributed processes. The notion of process has been widely used in operating system structures[5,6] in order to provide a modular representation of autonomous, event-driven computational tasks. Specification and implementation of protocols—well-defined conventions by which processes communicate—has allowed resource sharing to occur in a non-homogeneous environment. A process might utilize a remote resource such as a disk file, for example, by transmitting a prescribed command to a remote process which interprets and carries out the hypothetical command: "read the tenth record of disk file XYZ and transmit back its contents."

A layered structure of protocols has evolved to make possible network-wide sharing of ARPANET resources. The Host-Host protocol rests at the foundation of this structure, providing a mechanism by which processes in the network may communicate.[7] A number of higher level protocols make use of Host-Host protocol to perform function-oriented tasks.[8] For example, the Telnet protocol provides terminal access to remote interactive systems on the network, and the File Transfer Protocol allows files to be copied from one site to another.

The standard ARPANET Host-Host protocol creates inter-process communication (IPC) channels, or "connections," at the request of Host processes, through an exchange of special control messages between the Host operating systems. In general, this facility has been provided by the implementation of a set of procedures, collectively referred to as a Network Control Program, or NCP, which provides primitives for creation, control of data flow, and destruction of connections. An excellent survey of techniques used to implement the NCPs of various ARPANET Host systems has been given by Postel.[9]

Two major inferences regarding the desirable characteristics of Host operating system structure may be drawn from the ARPANET's evolution. First, the real-time event-driven nature of Host-IMP and Host-Host interaction requires some form of multiprogramming (i.e., multiple-process) capability in the Hosts. A Host system which supports terminal access to a network, for example, might utilize a process for each terminal. Each of these processes waits until a network message is received for terminal display or until a key-code is received for transmission to a remote system.

A second significant structural requirement of Host operating systems is a mechanism for communication

* This work was supported by the Advanced Research Projects Agency, Department of Defense, through Contract Number N00014-73-C-0221, administered by the Office of Naval Research.

between processes residing within a given Host ("local" processes) and processes residing in other Hosts on the network ("remote" processes). This inter-process communication (IPC) facility is achieved by the transmission of messages between Hosts according to an agreed-upon protocol. Implementation of network-wide IPC mechanisms (such as those embodied by the ARPANET Host-Host protocol) is greatly facilitated by the presence of internal mechanisms for IPC within each Host system.[10] Systems which lack these capabilities force the NCP to accept this responsibility; this is usually a fairly major implementation task when processes reside in different protected regions or address spaces.

Another important decision that must be made relates to the way in which the NCP is included in an operating system. The NCP may be embedded in the Host's file system, allowing network "connections" to be created and data transfers to occur in the same fashion that files are opened and read/written. This greatly facilitates the implementation of higher level protocols (e.g., file transfer) because it enables the standard file system primitives to be used for data transfer on network connections. Such an approach is practical when network protocol software is included as an integral part of the development of an operating system or when significant modification to an existing operating system may be tolerated. A disadvantage is that the file system must be modified when changes occur to low-level Host-Host protocols.

To simplify implementation and maintenance, it is desirable that an NCP run within a normal "user" job under the system. If system-wide IPC facilities are non-existent, this technique is feasible only when: (1) it is possible for the user job to usurp control on certain system calls which are issued by other user jobs, or (2) the scope of interaction with network resources is limited to a set of processes within a job, rather than globally available to all jobs. The first case makes assumptions about the protection structure which exists in the system, and is usually impractical when jobs occupy mutually exclusive protected memory spaces. The second case is feasible when the Host's sole function in the network is the management of resources which are allocated to the NCP "user" job; this technique might be used, for example, to provide access to a large data base. In both of the above cases it is necessary that the NCP implement an inter-process communication mechanism.

When a robust IPC facility is provided by an existing Host system it is possible to allow processes within the Host to communicate on a network-wide basis with minimal system modification. In this case, an NCP may exist as a user job, making use of the operating system's IPC facility to accept commands from the Host's processes (such as requests to open connections) and to handle the data transfer between local and remote processes.

Designers of Host operating systems for packet-switching networks must be sure that the chosen architecture provides sufficient flexibility. This is exemplified by the evolution of an ARPANET standard Host-Host protocol as well as special-purpose protocols for inter-network communication and packetized speech transmission. In some systems additional flexibility has been obtained by system calls which allow processes to intercept certain arriving messages and to transmit network messages directly, rather than forcing all network communication to occur by means of a standard Host-Host protocol.

HOST IPC MECHANISMS

Techniques for inter-process communication and synchronization in multiprogramming systems have received a good deal of attention.[6,11] There tend to be two strategies for implementing IPC systems. The first of these, like the telephone system, has required the establishment of a connection, or logical data path, before data may be transmitted between processes. In the distributed environment this strategy entails the utilization of special-purpose control messages which establish a name and control the data flow for the connection. The second approach has shunned the notion of prolonged connections, and performs the transfer of messages between processes whenever they mutually agree to communicate (e.g., process A requests to receive from process B while process B requests to send to process A). Control information (such as acknowledgment of messages received) is effectively embedded in each message exchanged between the Hosts. Walden has proposed such a connection-free mechanism[12] for inter-process communication within a packet-switching network. Metcalfe has also discussed the possibility of connection-free protocols.[13] In fact, it is becoming clear that the distinction should not actually be between connection-based protocols vs. connection-free protocols. The proper distinction to make is between protocols based on transmission of a single indefinitely long bit stream (starting when a connection is opened and ending when a connection is closed) vs. a stream of discrete messages (which may still require some connection-like control information). The message-based strategy is less sensitive to transmission errors which might occur in the communication subnetwork; the combination of control and data information within each transmitted message, for example, reduces the possibility of inconsistencies arising from the loss of a message. In addition, the message-based scheme requires minimal explicit connection setup, in some cases eliminating it entirely. Such an approach is being investigated in the design of Host-Host protocol techniques for interconnected packet-switching networks.[14]

There are various means by which IPC techniques have been implemented in operating systems. Some systems make use of a shared area of storage to pass messages between processes, utilizing the system's process synchronization techniques to announce the existence and acknowledgment of these messages. In other cases there exist special system calls (e.g., SEND, RECEIVE) which transfer data from a specified sender's buffer address to a matching receiver's buffer address. Akkoyunlu, Bernstein, and Schantz[15] have proposed a system (SBS) which supports inter-process communication and file I/O activities in a unified fashion. This approach has the distinct advantage of allowing processes to access data files stored within a remote system in the same way they would if the files were stored locally.

A similar IPC mechanism has been implemented in the ELF system[16] by means of the I/O primitives provided by the operating system. This technique enables the NCP to be included in the system as a user job, and all communication with the ARPANET occurs via this IPC structure. IPC occurs by means of a set of rendezvous-points, or ports, which appear identical to I/O devices except for differences in name. Processes may agree to communicate on a predesignated port, or may use a pair of such ports to exchange port numbers.

The ELF operating system provides a multiprogramming environment which allows creation and destruction of processes. Each process is named by means of a process-ID, owns an associated linked list structure called an event queue, and may be in one of two states: ready or blocked.

Processes synchronize by means of short (24-bit) messages which signal the occurrence of events. This is implemented by means of the following system primitives:

(1) SIGNAL (process-ID, event message)
(2) WAIT.

The SIGNAL primitive adds an event message to the event queue of process-ID; the SIGNALled process is placed in the ready state. The WAIT primitive tests the event queue of the active process, and places the process in the blocked state if the event queue is null; otherwise WAIT removes the first event message from the event queue and returns to the caller with the event message and the signalling process-ID. WAIT thus blocks the active process until an event message is placed on its event queue.

A process transmits data to another process by means of the primitive:

WRITE (port, mode, addr, count, event message),

in which port is the port name, mode denotes a stream or record-oriented transfer, addr is the address of data to be transmitted, count is the number of bytes for transfer, and event is an event message which the process wishes to receive (from WAIT) when the message has been sent. In record mode transfers the process is signalled when one or more bytes are taken by a receiver; in stream mode transfers the process is signalled when all bytes of the message are taken.

A process receives messages from another process by means of the primitive:

READ (port, mode, addr, count, event message).

All arguments are identical to those of the WRITE primitive. In record mode the receiving process is signalled when one or more bytes are placed in its input buffer; in stream mode the receiver is signalled only when all of the requested bytes have been placed in its input buffer. This allows a receiving process to reserve a large input buffer and wake-up when any data has been placed in its buffer (as is the case for a process awaiting input from the network and destined to be displayed on the user's terminal). Processes issuing WRITE or READ requests cause entries to be placed on a queue for the specified port; entries are removed from the queue when matching WRITEs and READs occur, and the appropriate transfer conditions (i.e., record or stream) are satisfied.

An additional primitive is provided to aid processes (e.g., and NCP) in gauging their allocation of buffer storage.

STAT (port)

returns the number of bytes which are queued to be written or read on the specified port. Special consideration is given when the requested count is specified as 0. In this case the WRITEing or READing process is signalled if there is a matching request on the specified port; the state of the port is unaffected in this case, and the process may then issue a normal WRITE or READ to transfer the pending data. This enables a process to wait for a matching request without locking up an input buffer for an unknown period of time.

OTHER HOST FACILITIES

Thus far we have dealt primarily with facilities for inter-process communication among Hosts in a distributed network. The capability of the Host as a viable network resource, however, depends heavily on services available in the Host operating system. In many cases this involves more than an implementation of protocols, and requires significant augmentation of Host facilities. Hosts providing interactive services, for example, must allow network access to occur in a fashion which is compatible with local terminal I/O. Many time-shared operating systems utilize a dedicated "logger" process which awaits the activation of previously dormant terminals. A mechanism is required to enable notification of the logger process when a network port becomes assigned on behalf of a user at a remote terminal. In a system which treats files and network connections uniformly this may be achieved by means of a system primitive which assigns a pair of files (i.e., the network connections) as controlling input and output data streams for the logger.

Batch-oriented service Hosts require the ability to redirect input and output files to network ports. This task is facilitated in systems which support remote job entry, allowing the set of network ports to be associated with a pseudo remote job entry terminal.

Host operating systems supporting file transfer must supply a flexible set of primitives for the allocation, updating, deletion, and directory maintenance of the file system.

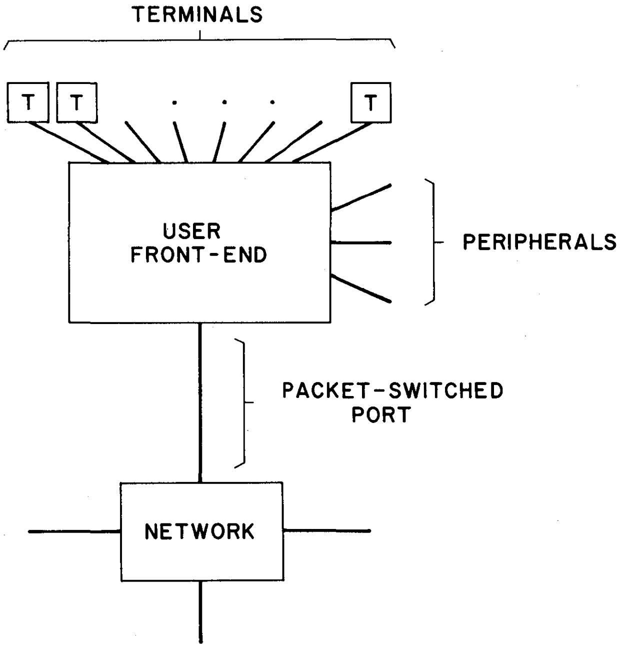


Figure 1—Typical user front-end system

In addition, a means of user authentication is usually required to provide an access control and accounting mechanism for the Host's resources.

FRONT-END SYSTEMS

In a number of cases it has been desirable to minimize the changes to a Host operating system when adding a preexisting Host to the network. In other cases, it has been desirable for reasons of reliability, flexibility, or increased Host system performance, to clearly separate the network functions from the Host. These two cases have frequently been handled by the addition of a front-end system as an interface between the network and the Host. Front-end Hosts are usually implemented by means of small computers utilizing operating systems which support network protocols as well as various terminal and peripheral handlers.

The most common application of front-end systems in the ARPANET results from the need for user terminal and peripheral access to network computing resources. The TIP,[17] ANTS,[18] and ELF[16] systems are examples of such user front-ends, shown in Figure 1. In the above systems, for example, users at terminals may LOGIN at the front-end and use the Telnet protocol to connect to various server sites on the network. Data may be transferred to or from attached peripheral devices (line printers, magnetic tape units, disk drives) by means of a file transfer protocol.

A second type of front-end system facilitates the attachment of a computing resource (server Host) to a network. This approach aims to relieve the server Host of network communication tasks, such as those required to support Host-Host protocols, and is desirable when software modifications to an existing Host system are prohibited.

The structural requirements of the operating system in the server front-end are similar to those in the user front-end. (In fact, systems may serve both as a user and as a server front-end.) Figure 2 illustrates the means of interconnection of server and server front-end. Processes in the server front-end respond to (user) requests from the network, and provide access to the server via a number of server/front-end ports (hard-wired connections). For example, a front-end system might be connected to a number of terminal interface ports belonging to a server system which supports interactive terminal access. When a network message which requests connection to the server is received by the front-end, a process is created in the front-end; this process then allocates an unused server/front-end port and initiates requests for (full-duplex) data transfer between the remote network process and the server front-end port. This approach allows the

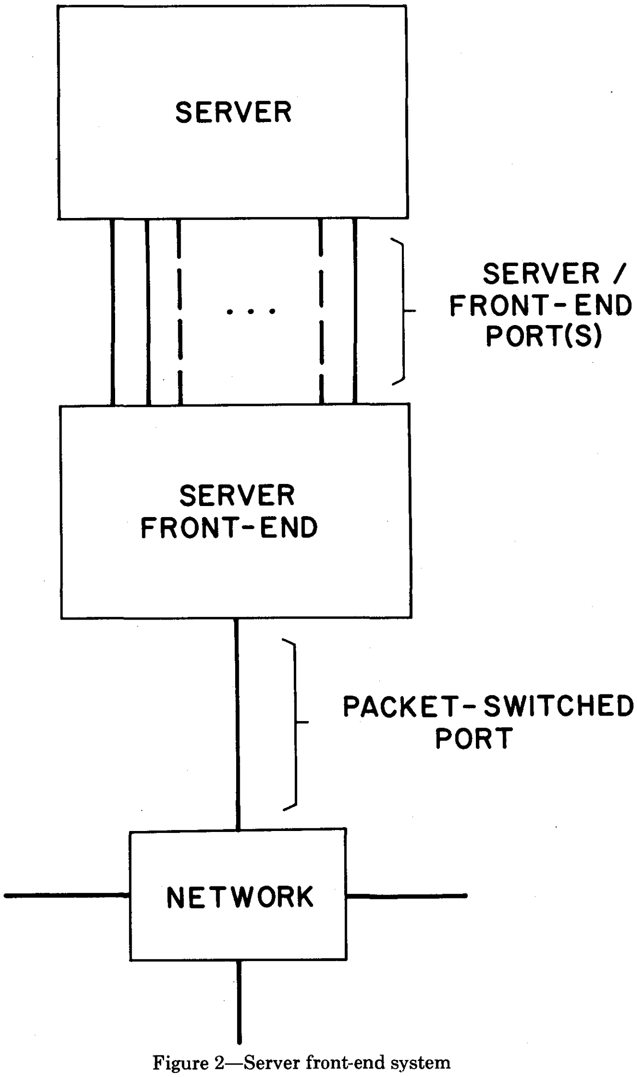


Figure 2—Server front-end system

server to be accessed from the network with no change to server software; however, it lacks generality, and server capabilities are limited to those functions associated with terminals. The server front-end model also applies to batch-oriented systems, in which case the style of connection between the server and the server front-end might resemble a set of card readers or printers.

When software additions are feasible in the server, a server/front-end protocol may be utilized to permit communication over a single hard-wired connection. As in the case of NCP implementations described earlier, however, general access to network resources by processes within the server requires some form of server IPC capability.

Interestingly, in some cases, functions may be performed by a large Host on behalf of a small front-end. For example, the means of providing control of access to a network requires access to a data base of user names, passwords, account numbers, and so forth. This may be accomplished by allowing the front-end system to simultaneously request connection (broadcast) to a number of "server" Hosts, accepting the first successful completion of a connection, and then requesting the attached server to perform the user authentication task; such a cooperative technique is in use between the TIP and TENEX RSEXEC[19] systems on the ARPANET. This type of interaction between Host systems is an example of automated resource sharing.

AUTOMATED RESOURCE SHARING

Host operating systems in the network environment may be structured to allow automatic utilization of resources in other Hosts. This subject has generated considerable interest because there is need for distributed data bases and load sharing in the network environment. As the size of network user communities expands, it becomes increasingly important to automate the allocation of processing and storage resources to allow their widespread and efficient use, while reducing problems faced by unsophisticated users. The realization of these techniques involves a diversion from the traditional image of centralized operating system structures; it involves the management of resources which are distributed, as opposed to centralized, by means of the coordination of distributed processes. This coordination requires within each centralized system a well-defined protocol and a flexible set of facilities to enable the processes to reliably carry out the management of network-wide resources.

Facilities provided by such a distributed operating system make possible common file naming schemes which are global to the network, thereby freeing a user of the responsibility of remembering a particular file's location. In addition, a network-wide user (directory) naming convention may be established. Tools for user-user communication, such as mail facilities, direct terminal intercommunication, and conferencing are needed on a network-wide basis. These capabilities will become increasingly desirable as the number of widely-dispersed service systems and users requiring access to shared data bases increases. Techniques are required for automatic archiving of network data bases; similarly, there is need for automatic retrieval of files (or portions thereof) upon reference, in much the same way as information flows between levels of storage in current centralized hierarchical storage systems. Of course, solutions to these problems raise a number of complex data management, accounting and security issues.

Development of the TENEX RSEXEC[19] System has been a step in this direction, providing users with a unified network file directory structure and terminal-terminal communication capabilities. The associated RSEXEC protocol allows the involvement of a number of TENEX sites, and allows other non-TENEX systems to participate. The success of this effort has been made possible in part by the characteristics of the TENEX operating system, which provides a tree-structured process and virtual memory capability.[20]

The development of distributed operating system structures requires a number of support facilities within each centralized system. A "system within a system" approach is desirable in which the resource-sharing processes utilize system primitives available in the parent operating system and create sub-processes which carry out remotely requested tasks. The creator processes intercept system calls issued by the sub-processes, providing them with a different virtual programming environment from that provided by the parent operating system. For example, a subprocess may access files in a network-wide directory structure by means of what it thinks are standard system calls.

Load-sharing techniques are aimed at an allocation of resources which provides distribution of load or assignment of processing tasks to the most appropriate server sites. For example, it is possible to allow tasks to be cooperatively carried out (simultaneously) by a number of servers; a protocol is being designed and implemented which allows processes to call procedures which execute on differing machines.[21] Basic problems are encountered in attempts at dynamic distribution of load: programs may depend on locally available data bases or on hardware or software peculiarities of a particular system.[22] These problems may be relieved by establishing a "standard" set of programs (e.g. editors, text-formatting programs, compilers) and conventions which guarantee compatibility with the virtual programming environment provided by the resource-sharing processes. The practicality of performing this type of load sharing, however, hinges on the existence of methods of accessing remotely distributed data.

SUMMARY

This paper has discussed structural characteristics which are required in Host operating systems for a packet-switching network; the need for flexible mechanisms for inter-process communication in the network Host (along with an example of an IPC mechanism for an ARPANET

Host System); the function and structure of "front-end" Host systems as network user and server interfaces; and finally, the need for development of automated techniques for managing resources in the distributed environment. These techniques will eventually provide capabilities for widespread access to shared data bases and new services for user-user communication. Their development requires well-structured centralized operating systems which serve as building blocks in the framework of a network-wide distributed operating system.

ACKNOWLEDGMENTS

The author wishes to thank R. Brooks, J. Malman, J. Miller, M. Retz, and D. Walden who assisted in the development of this manuscript.

REFERENCES

1. Roberts, L. G., F. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *Proceedings of AFIPS SJCC,* 1970, pp. 543-549.
2. Kahn, R. E., "Resource Sharing Computer Communications Networks *Proceedings of the IEEE,*" Vol. 60, No. 11, November 1972, pp. 1397-1407.
3. Pouzin, L., "Presentation and Major Design Aspects of the CYCLADES Computer Network," *Proc. 3rd. Data Communications Symp.,* 1973, pp. 80-88.
4. Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther, D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," *Proceedings of AFIPS SJCC,* 1970, Vol. 36, pp. 551-567.
5. Dijkstra, E. W., "The Structure of the "THE"—Multiprogramming System," *Communications of the ACM,* Vol. 11, No. 5, May 1968, pp. 341-346.
6. Horning, J. J., B. Randell, *"Process Structuring," ACM Computing Surveys,* Vol. 5, No. 2, March 1973, pp. 5-30.
7. McKenzie, A. A., *HOST/HOST Protocol for the ARPA Network,* National Technical Information Service, AD757680.
8. Crocker, S. D., R. M. Metcalfe, J. B. Postel, J. F. Heafner, "Function-Oriented Protocols for the ARPA Computer Network," *Proceedings of AFIPS SJCC,* 1972, Vol. 40, pp. 271-279.
9. Postel, J. B., *Survey of Network Control Programs in the ARPA Computer Network,* MITRE Technical Report #6722, October 1974, Revision 1, 89 pp.
10. Metcalfe, R. M., "Strategies for Operating Systems In Computer Networks," *Proceedings of the ACM Annual Conference,* August 1972, pp. 278-281.
11. Spier, M., E. Organick, "The MULTICS Interprocess Communication Facility," *Proceedings ACM Second Symposium on Operating System Principles,* Princeton University, October 20-22, 1969, pp. 83-91.
12. Walden, D. C., "A System for Interprocess Communication in a Resource Sharing Computer Network," *Communications of the ACM,* Vol. 15, No. 4, April 1972, pp. 221-230.
13. Metcalfe, R. M., *Packet Communication,* MIT Report, MAC TR-114, December 1973.
14. Cerf, V. G., R. E. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Transactions on Communications,* Vol. COM-22, No. 5, May 1974, pp. 637-648.
15. Akkoyunlu, E., A. Bernstein, R. Schantz, "Interprocess Communication Facilities for Network Operating Systems," *Computer,* June 1974, pp. 46-55.
16. Retz, D., J. Miller, J. McClurg, B. Schafer, *ELF System Programmer's Guide,* September 1974, Speech Communications Research Lab, Inc., Santa Barbara, California.
17. Ornstein, S. M., F. E. Heart, W. R. Crowther, H. K. Rising, S. B. Russell, A. Michel, "The Terminal IMP for the ARPA Computer Network," *Proceedings of AFIPS SJCC,* 1972, Vol. 40, pp. 243-254.
18. Bouknight, W. J., S. Denenberg, *ANTS—A New Approach to Accessing the ARPA Network,* University of Illinois Report CAC No. 47, July 1972.
19. Thomas, R. H., "A Resource Sharing Executive for the ARPANET," *Proceedings of AFIPS NCC,* 1973, pp. 155-163.
20. Bolt Beranek and Newman Inc., Computer Science Division, *TENEX JSYS MANUAL—A Manual of TENEX Monitor Calls.*
21. White, J. E., *Procedure Call Protocol Specification,* November 1974, Augmentation Research Center, Stanford Research Institute.
22. Dennis, J. B., "A Position Paper on Computing and Communications," *Communications of the ACM,* Vol. 11, No. 5, May 1968, pp. 370-377.

Issues in packet switching network design*

by W. R. CROWTHER, F. E. HEART, A. A. McKENZIE, J. M. McQUILLAN, and D. C. WALDEN

Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

INTRODUCTION

The goals of this paper are to identify several of the key design choices that must be made in specifying a packet-switching network and to provide some insight in each area. Through our involvement in the design, evolution, and operation of the ARPA Network over the last five years (and our consulting in the design of several other networks), we have learned to appreciate both the opportunities and the hazards of this new technical domain.

The last year or so has seen a sudden increase in the number of packet-switching networks under consideration worldwide. It is natural that these networks try to improve on the example of the ARPA Network, and therefore that they contain many features different from those of the ARPA Network. We recognize that networks must be designed differently to meet different requirements; nevertheless, we think that it is easy to overlook important aspects of performance, reliability, or cost. It is vital that these issues be adequately understood in the development of very large practical networks—common user systems for hundreds or thousands of Hosts—since the penalties for error are correspondingly great.

Some brief definitions are needed to isolate the kind of computer network under consideration here:

Nodes. The nodes of the network are real-time computers, with limited storage and processing resources, which perform the basic packet-switching functions.

Hosts. The Hosts of the network are the computers, connected to nodes, which are the providers and users of the network services.

Lines. The lines of the network are some type of communications circuit of relatively high bandwidth and reasonably low error rate.

Connectivity. We assume a general, distributed topology in which each node can have multiple paths to other nodes, but not necessarily to all other nodes. Simple networks such as stars or rings are degenerate cases of the general topology we consider.

Message. The unit of data exchanged between source Host and destination Host.

Packet. The unit of data exchanged between adjacent nodes.

Acknowledgment. A piece of control information returned to a source to indicate successful receipt of a packet or message. A packet acknowledgment may be returned from an adjacent node to indicate successful receipt of a packet; a message acknowledgment may be returned from the destination to the source to indicate successful receipt of a message.

Store and Forward Subnetwork. The node stores a copy of a packet when it receives one, forwards it to an adjacent node, and discards its copy only on receipt of an acknowledgment from the adjacent node, a total storage interval of much less than a second.

Packet Switching. The nodes forward packets from many sources to many destinations along the same line, multiplexing the use of the line at a high rate.

Routing Algorithm. The procedure which the nodes use to determine which of the several possible paths through the network will be taken by a packet.

Node-Node Transmission Procedures. The set of procedures governing the flow of packets between adjacent nodes.

Source-Destination Transmission Procedures. The set of procedures governing the flow of messages between source node and destination node.

Host-Node Transmission Procedures. The set of procedures governing the flow of information between a Host and the node to which that Host is directly connected.

Host-Host Transmission Procedures. The set of procedures governing the flow of information between the source Host and the destination Host.

Within the class of network under consideration, there are already several operational networks and many network designs. The ARPA Network[1] is made up of over fifty node computers called IMPs and over seventy Hosts. The Cyclades Network[2] is a French network consisting of about six nodes and about two Hosts per node. The Societe Internationale de Telecommunication Aeronautique (SITA) Network[3] connects centers in eight or so cities mostly in Europe. The European Informatics Network (EIN),[4] also known as Cost-11, is currently in a design stage and will be a network interconnecting about six computers in several Common Market countries. Some other packet-switching network designs include: Autodin II,[5] NPL,[6] PCI,[7] RCP,[8] and Telenet.[7]

* This work was supported under Advanced Research Projects Agency Contracts DAHC15-69-C-0179 and F08606-73-C-0027.

Some of the more obvious differences among these networks can be cited briefly. The ARPA Network splits messages into packets up to 1000 bits long; the other networks have 2000-bit packets and no multipacket messages. Hosts connect to a single node in the ARPA Network and SITA; multiple connections are possible in Cyclades and EIN. Dynamic routing is used in the ARPA Network and EIN; a different adaptive method is used in SITA; fixed routing is presently used in Cyclades. The ARPA Network delivers messages to the destination Host in the same sequence as it accepts them from the source Host; Cyclades does not; in EIN it is optional. Clearly, many of the design choices made in these networks are in conflict with each other. The resolution of these conflicts is essential if balanced, high-performance networks are to be planned and built, particularly since many future designs will be intended for larger, less experimental, and more complex networks.

FUNDAMENTAL ISSUES

In this section we define what we believe are fundamental properties and requirements of packet-switching networks and what we believe are the fundamental criteria for measuring network performance.

Network properties and requirements

We begin by giving the properties central to packet-switching network design. The key assumption here is that the *packet processing algorithms* (acknowledgment/retransmission strategies used to control transmission over noisy circuits, routing, etc.) result in a virtual network path between the Hosts with the following characteristics:

a. Finite, fluctuating delay—A result of the basic line bandwidth, speed of light delays, queueing in the nodes, line errors, etc.
b. Finite, fluctuating bandwidth—A result of network overhead, line errors, use of the network by many sources, etc.
c. Finite packet error rate (duplicate or lost packets)—A result of the acknowledgment system in any store-and-forward discipline (this is a different use of the term "error rate" than in traditional telephony). Duplicate packets are caused when a node goes down after receiving a packet and forwarding it without having sent the acknowledgment. The previous node then generates a duplicate with its retransmission of the packet. Packets are lost when a node goes down after receiving a packet and acknowledging it before the successful transmission of the packet to the next node. An attempt to prevent lost and duplicate packets must fail as there is a tradeoff between minimizing duplicate packets and minimizing lost packets. If the nodes avoid duplication of packets whenever possible, more packets are lost. Conversely, if the nodes retransmit whenever packets may be lost, more packets are duplicated.
d. Disordering of packets—A property of the acknowledgment and routing algorithms.

These four properties describe what we term the *store-and-forward subnetwork.*

There are also two basic problems to be solved by the source and destination* in the virtual path described above:

e. Finite storage—A property of the nodes.
f. Differing source and destination bandwidths—Largely a property of the Hosts.

A slightly different treatment of this subject can be found in Reference 9.

The fundamental requirements for packet-switching networks are dictated by the six properties enumerated above. These requirements include:

a. Buffering—Buffering is required because it is generally necessary to send multiple data units on a communications path before receiving an acknowledgment. Because of the *finite delay* of the network, it may be desirable to have buffering for multiple packets in flight between source and destination in order to increase throughput. That is, a system without adequate buffering may have unacceptably low throughput due to long delays waiting for acknowledgment between transmissions.

b. Pipelining—The *finite bandwidth* of the network may necessitate the pipelining of each message flowing through the network by breaking it up into packets in order to decrease delay. The bandwidth of the circuits may be low enough so that forwarding the entire message at each node in the path results in excessive delay. By breaking the message into packets, the nodes are able to forward the first packet of the message through the network ahead of the later ones. For a message of P packets and a path of H hops, the delay is proportional to $P+H-1$ instead of $P * H$, where the proportionality constant is the packet length divided by the transmission rate.**

c. Error Control—The node-to-node packet processing algorithm must exercise error control, with an acknowledgment system in order to deal with the *finite packet error rate* of the circuits. It must also detect when a circuit becomes unusable, and when to begin to use it again. In the source-to-destination message processing algorithm, the destination may need to exercise some controls to detect missing and duplicated messages or portions of messages, which would appear as incorrect data to the end user. Further, acknowledgments of message delivery or non-delivery may be useful, possibly to trigger retransmission. This mechanism in turn requires error control and retransmission itself, since the delivery reports can be lost

* The question of whether the source and destination nodes or the source and destination Hosts should solve these problems is addressed in a later section.

** See page 90 of Reference 9 for a derivation and more exact result.

or duplicated. The usual technique is to assign some unique number to identify each data unit and to time out unanswered units. The error correction mechanism is invoked infrequently, as it is needed only to recover from node or line failures.

d. Sequencing—Since *packet sequences can be received out of order,* the destination must use a sequence number technique of some form to deliver messages in correct order, and packets in order within messages, despite any scrambling effect that may take place while several messages are in transit. The sequencing mechanism is frequently invoked since it is needed to recover from line errors.

e. Storage allocation—The fact that *storage* in the nodes is *finite* means that both the packet processing and message processing algorithms must exercise control over its use. The storage may be allocated at either the sender or the receiver.

f. Flow Control—The *different source and destination data rates* may necessitate implicit or explicit flow control rules to prevent the network from becoming congested when the destination is slower than the source. These rules can be tied to the sequencing mechanism, with no more messages (packets) accepted after a certain number, or tied to the storage allocation technique, with no more messages (packets) accepted until a certain amount of storage is free, or the rules can be independent of these features.

In satisfying the above six requirements, the algorithm often exercises contention resolution rules to allocate resources among several users. The twin problems of any such facility are:

- fairness—resources should be used by all users fairly;
- deadlock prevention—resources must be allocated so as to avoid deadlocks.

We have also come to believe that it is essential to have a reset mechanism to unlock "impossible" deadlocks and other conditions that may result from hardware or software failures.

Network performance goals

Packet-switching communications systems have two fundamental goals in the processing of data—low delay and high throughput. Each message should be handled with a minimum of waiting time, and the total flow of data should be as large as possible. The difference between low delay and high throughput is important. What the network user wants is the completion of his data transmission in the shortest possible time. The time between transmission of the first bit and delivery of the first bit is a function of network delay, while the time between delivery of the first bit and delivery of the last bit is a function of network throughput. For interactive users with short messages, low delay is more important, since there are few bits per message. For the transfer of long data files, high throughput is more important.

There is a fundamental tradeoff between low delay and high throughput, as is readily apparent in considering some of the mechanisms used to accomplish each goal. For low delay, a small packet size is necessary to cut transmission time, to improve the pipelining characteristics, and to shorten queueing latency at each node; furthermore, short queues are desirable. For high throughput, a large packet size is necessary to decrease the circuit overhead in bits per second and the processing overhead per bit. That is, long packets increase the effective circuit bandwidth and nodal processing bandwidth. Also, long queues may be necessary to provide sufficient buffering for full circuit utilization. Therefore, the network may need to employ separate mechanisms if it is to provide low delay for some users and high throughput for others.

To these two goals one must add two other equally important goals, which apply to message processing and to the operation of the network as a whole. First, the network should be cost-effective. Individual message service should have a reasonable cost as measured in terms of utilization of network resources; further, the network facilities, primarily the node computers and the circuits, should be utilized in a cost-effective way. Secondly, the network should be reliable. Messages accepted by the network should be delivered to the destination with a high probability of success. And the network as a whole should be a robust computer communications service, fault-tolerant, and able to function in the face of node or circuit failures.

In summary, we believe that delay, throughput, reliability, and cost are the four criteria upon which packet-switching network designs should be evaluated and compared. Further, it is the combined performance in all four areas which counts. For instance, poor delay and throughput characteristics may be too big a price to pay for "perfect" reliability.

Key design choices

We believe there are three major areas in which the key choices must be made in designing a packet-switching network. First, there is network hardware design, including the node computer, the network circuits, the Host-to-node connections, and overall connectivity. Second, there is store-and-forward subnetwork software design, primarily the routing algorithm and the node-to-node transmission procedures. Third, there is source-to-destination software design, which encompasses end-to-end transmission procedures and the division of responsibility between Hosts and nodes.* These topics are covered in the following sections.

* There are strong interactions between the topics discussed in the second and third areas. The end-to-end traffic requirements of a specific user can only be met if the store-and-forward subnetwork has mechanisms which act in concert with the source-to-destination mechanisms to provide the required performance. Discussion of this interaction, an important consideration in packet-switching network design, is beyond the scope of this paper.

NETWORK HARDWARE DESIGN

In this section we outline some of the design issues associated with the choice of the node computer, the network circuits, the Host-to-node connections, and overall connectivity. Since the factors affecting these choices change rapidly with the introduction of new technology, we discuss only general observations and design questions.

The node computer

The architecture of the node computer is related to several other network design parameters, as detailed below.

Processor

The speed of the processor is important in determining the throughput rates possible in the network. The store-and-forward processing bandwidth of the processor can be computed by counting instructions in the inner loop (see Reference 10 for an example). The source-to-destination processing bandwidth can be calculated in a similar fashion. These rates should be high enough so that the entire bandwidth of the network lines can be used, i.e., so that the node is not a bottleneck. It has been our experience that the speed of the processor and memory is the main factor in this bandwidth calculation; complex or specialized instruction sets are not valuable because simple instructions make up most of the node program.

A different aspect of the node computer which can also affect throughput is its responsiveness. Because circuits are synchronous devices, they require service with very tight time requirements. If the node does not notice that input has completed on a given circuit, and does not prepare for a new input within a given time, the next input arriving on that circuit will be lost. Likewise on output, the node must be responsive in order to keep the circuits fully loaded. This requirement suggests that some form of interrupt system[1] or high-speed polling device[11] is necessary to keep response latency low, and that the overhead of an operating system and task scheduler and dispatcher may be prohibitive. Finally, we note that the amount of time required by the node to process input and output is most critical in determining the *minimum* packet size, since it is with packets of this size that the highest packet arrival and departure rates (and thus processing requirements) can be observed. Of course, data buffering in the device interfaces can partially alleviate these problems.

Memory

The speed of memory may be a major determinant of processor speed, thus affecting the node bandwidth. An equally important consideration is memory speed for I/O transfers, since the node's overall bandwidth results from a division of total memory bandwidth based on some processing time for a given amount of I/O time. First, there is the question of whether the I/O transfers act in a cycle-stealing fashion to slow the processor or whether memory is effectively multi-ported to allow concurrent use. Then there is the issue of contention for memory among the various synchronous I/O devices. In a worst-case scenario, it is possible for all the I/O devices to request a memory transfer at the same instant, which keeps memory continuously busy for some time interval. A key design parameter is the ratio of this time to the available data buffering time of the least tolerant I/O device. This ratio should be less than one, and may therefore determine how much I/O can be connected to the node.

The size of the memory, naturally, is another key parameter. It has been our experience[1,10] that the program and associated data structures take up the majority of storage in the node. The remainder of memory is devoted to buffering of two kinds: packet buffering between adjacent nodes, and message buffering between source and destination nodes. These requirements can be calculated quite simply in each case as the product of the maximum data rate to be supported times the round trip time (for a returning acknowledgment). In large networks it may be necessary to rely on sophisticated compression techniques to ensure that tables for the routing algorithm, the source-to-destination transmission procedures, and so on, do not require excessive storage.

I/O

The speed of the I/O system has been touched upon above in relation to processor and memory bandwidth. Other factors worth noting are the internal constraints imposed by the I/O system itself—its delay and bandwidth. A different dimension, and one that we have found to be inadequately designed by most manufacturers, is the flexibility and extensibility of the I/O system. Most manufacturers supply only a limited range of I/O options (some of which may be too slow or too expensive to use). Further, only a limited number of each type can be connected. A packet-switching network node requires high performance from the I/O system, both in the number of connections and in their data rates.

General architecture

There are other factors to consider in evaluating or designing a node processor apart from performance in terms of bandwidth and delay. As we mentioned, extensibility in I/O is very important and comparatively rare; it is more common to find memory systems which can be expanded. Processor systems which can be expanded are not at all common, and yet processor bandwidth may be the limiting factor in some node configurations. Without a modular approach allowing processing, memory and I/O growth, the cost of the node computer can be quite high due to large step functions in component cost.

Another aspect of node computer architecture is its reliability, particularly for large systems with many lines and Hosts. A failure of such a system has a large impact on network performance. We have studied these issues of performance, cost, and reliability of node computers in a packet-switching network, and have developed, under ARPA sponsorship, a new approach to this problem. Our computer, called the Pluribus, is a multiprocessor made up of minicomputers with modular processors, memory, and I/0 components, and a distributed bus architecture to interconnect them.[11] Because of its specially designed hardware and software features,[12] it promises to be a highly reliable system. We point out that many of these issues of performance, cost, and reliability could become critically important in very large networks serving thousands of Hosts and terminals.

We also note that there are so many stringent technical constraints on the computer that a choice made on other grounds (e.g., expediency, politics), as is common, is particularly unfortunate.

The network circuits

We next consider some of the important characteristics of the circuits used in the network.

Bandwidth

The bandwidth of the network circuits is likely to be their most important characteristic. It defines the traffic-carrying capacity of the network, both in the aggregate and between any given source and destination. What is less obvious is that the bandwidth (and hence the time to clock a packet out onto the line) may be the main factor determining the transit delays in the network. The minimum delay through the network depends mainly on circuit rates and lengths, and additional delays are largely accounted for by queueing delay, which is directly proportional to circuit bandwidth. These two factors lead to the general observation that the faster the network lines, the longer the packet should be, since long packets have less overhead and permit higher throughput, while the added delay due to length is less important at high circuits rates. In addition, more packet and message buffering is required when higher speed circuits are used.

Delay

The major effect of circuits with appreciable delay is that they require more buffering in the nodes to keep them fully loaded. That is, the node must maintain more packets in flight at once over a circuit with longer delay. This effect may be so large (a circuit using a satellite has a delay of a quarter of a second) as to require significantly more memory in the nodes.[10] This memory is needed at the nodes connected to the circuit to permit sufficient packet buffering for node-to-node transmission using the circuit. The subtle point is that additional buffering is also required at all nodes in the network that may need to maintain high source-to-destination rates over network paths which include this circuit. If they are to provide maximum throughput, they need sufficient message buffering to keep the entire network path fully loaded.

Reliability

Traditionally, the telephone carriers have quoted error rates in the following manner: "No more than an average of 1 bit in 10^6 bits in error." This definition is not entirely adequate for packet switching, though it may be for continuous transmission. For packet switching, the average bit error rate is less interesting than the average packet error rate (packets with one or more bits in error). For example, ten bits in error in every tenth packet is a 10 percent packet error rate, while one bit in error in every packet is a 100 percent packet error rate, yet the two cases have the same bit error rate.

An example of an acceptable statement of error performance would be as follows:

The circuit operates in two modes. Mode 1: no continuous sequence of packet errors longer than two seconds, with the average packet error rate less than one in a thousand. Mode 2: a continuous sequence of errors longer than two seconds with the following frequency distribution:

> 2 seconds	no more often than once per day
> 1 minute	no more often than once per week
>15 minutes	no more often than once per month
> 1 hour	no more often than once per 3 months
> 6 hours	no more often than once per year
> 1 day	never

While the figures above may seem too stringent in practice, the mode 1 bit error rate is actually quite lax compared to conventional standards. In any case, these are the kinds of behavior descriptions needed for intelligent design of packet-switching network error control procedures. Therefore, it is important that the carriers begin to provide such descriptions.

The packet error rate of a circuit has two main effects. First, if the rate is high enough, it can degrade the effective circuit bandwidth by forcing the retransmission of many packets. While this is basically a problem for the carrier to repair, the network nodes must recognize this condition and decide whether or not to continue to use the circuit. This is a tradeoff between reduced throughput with the circuit and increased delay and less network connectivity without it. Before the circuit can be used, it must be working in both directions for packets and for control information like routing and acknowledgments, and with a sufficiently low packet error rate.

The second effect of the error rate is present even for relatively low error rates. It is necessary to build a very good error-detection system so that the users of the network do not see errors more often than some specified extremely low frequency. That is, the network should detect enough errors so that the effective network error rate is at least an order of magnitude less than the Host error or failure rate. A usual technique here is a cyclic redundancy check on each packet. This checksum should be chosen carefully; to first order, its size does not depend on packet length* and it should be quite large, for example 24 bits for 50-Kbs lines and 32 bits for multi-megabit lines or lines with high error rates.

The Host-to-node connections

We examine the bandwidth and reliability of the Host connection to the network in the next two sections.

Bandwidth

The issues in choosing the bandwidth of the Host connections are similar to those for the network circuits. In addition to establishing an upper bound on the Hosts' throughput, the rate is also an important factor in delay. The delay to send or receive a long message over a relatively slow Host connection may be comparable in magnitude to the network round trip time. To eliminate this problem, and also to allow high peak throughput rates, the Host connection bandwidth should be as high as possible (within the limits of cost-effectiveness), even higher than the average Host throughput would indicate. By the same argument given above for packet size, a higher speed Host connection allows the use of a longer message with less overhead and Host processing per bit and therefore greater efficiency.

Reliability

The reliability of the Host connection is an important aspect of the network design; several points are worth noting. First, the connection should have a packet error rate which is at least as low as the network circuits. This can be accomplished by a highly reliable direct connection locally or by error-detection and retransmission. The use of error control procedures implies that the Host-node transmission procedures resemble the node-node transmission procedures which are discussed in a later section. Second, if the Host application requires extremely high reliability, a Host-to-Host data checksum and message sequence check are both useful for detecting infrequent network failures. Third, if the Host requires uninterrupted network service, and the Host is reliable enough itself to justify such service, multiple connections of the Host to various nodes can improve the availability of the network. This option complicates matters for the source-to-destination transmission procedures in the nodes (e.g., sequencing) since there may be more than one possible destination node serving the Host.

Overall connectivity

The subject of network topology is a complex one,[13] and we limit ourselves here to a few general observations. In practice, it seems that the connectivity of the nodes in the network should be relatively uniform. It is obvious that nodes with only a single line are to be avoided for reliability considerations but nodes with many circuits also present a reliability problem since they remove so much network connectivity when they are down. We also feel that the direction for future evolution of network geometries will be toward a "central office" kind of layout with relatively fewer nodes and with a high fan-in of nearby Hosts and terminals. This tendency will become more pronounced as higher reliability in the node computer becomes possible, even for large systems. One reason that we favor this approach is that a large node computer presents an increased opportunity for shared use of the node resources (processor and memory) among many different devices leading to a much more efficient and cost-effective implementation. This trend will mean that in the future, even more than now, a key cost of network topology will be the ultimate data connection to the user (Host or terminal), who may be far from the central office. Concentrators and multiplexors have been the traditional solution; in packet-switching networks, a small node computer could fill this function. In conclusion, we see flexibility and extensibility as two key requirements for the node computer. These factors together with increasing performance and fan-in requirements imply a very high reliability standard as well.

STORE-AND-FORWARD SUBNETWORK SOFTWARE DESIGN

We cover two major areas in our discussion of store-and-forward subnetwork software design, the routing algorithm and the node-to-node transmission procedures, both of which are packet-oriented and require no information about messages.

The routing algorithm

The fundamental step in designing a routing algorithm is the choice of the control regime to be used in the operation of the algorithm. *Non-adaptive* algorithms make no real attempt to adjust to changing network conditions; no

* Assuming that the probability of packet error is proportional to the product of packet length and bit error rate, the checksum length should be proportional to the log of the product of the desired time between undetected errors, the bit error rate, and the total bandwidth of all network circuits.

routing information is exchanged by the nodes, and no observations or measurements are made at individual nodes. *Centralized adaptive* algorithms utilize a central authority which dictates the routing decisions to the individual nodes in response to network changes. *Isolated adaptive* algorithms operate independently with each node making exclusive use of local data to adapt to changing conditions. *Distributed adaptive* algorithms utilize internode cooperation and the exchange of information to arrive at routing decisions.*

Non-adaptive algorithms

Under this heading come such techniques as fixed routing, fixed alternate routing, and random routing (also known as flooding or selective flooding).

Simple fixed routing is too unreliable to be considered in practice for networks of more than trivial size and complexity. Any time a single line or node fails, some nodes become unable to communicate with other nodes. In fact, networks utilizing fixed routing always assume manual updates (as necessary) to another fixed routing pattern. However, in practice this means that every routine network component failure becomes a catastrophe for operational personnel, every site spending frantic hours manually reconstructing routing tables.[15]

At their best, in the absence of network component failure, fixed routing algorithms are inefficient. While the routing tables can be fixed to be optimal for some traffic flow, fixed routing is inevitably inefficient to the extent that network traffic flows vary from the optimal traffic flow. Unreliability and inefficiency are also characteristic of two alternative techniques to fixed routing which fall under the heading of non-adaptive algorithms: fixed routing with fixed alternate routes and random routing.[14]

Non-adaptive algorithms are all extremely simple and can therefore be implemented at low cost. They are thus possibly suitable for hardware implementation, for theoretical analysis and for studying the effects of varying other network parameters and algorithms.

In conclusion, we do not recommend non-adaptive routing for most networks because it is unreliable and inefficient. Despite these drawbacks, many networks have been proposed or begun with non-adaptive routing, generally because it is simpler to implement and to understand. Perhaps this tendency will be reversed as more information about other routing techniques is published and as network technology generally grows more sophisticated.

Centralized adaptive algorithms

In a centralized adaptive algorithm, the nodes send the information needed to make a routing decision to a Routing Control Center (RCC) which dictates its decision back to the nodes for actual use. The advantages claimed for a centralized algorithm are: (a) the routing computation is simpler to understand than a non-centralized algorithm, and the computation itself can follow one of several well known algorithms, e.g.;[16] (b) the nodes are relieved of the burden and overhead of the routing computation; (c) more nearly optimal routing is possible because of the sophistication that is possible in a centralized algorithm; and (d) routing "loops" (a possible temporary property of distributed algorithms) can be avoided.

Unfortunately, the processor bandwidth utilization at the center is likely to be very heavy. The classical algorithms that a centralized approach might use generally run in time proportional to N^3 (where N is the number of nodes in the network), while their distributed counterparts can run (through parallel execution) in time proportional to N^2. While it may be a saving to remove computation from the nodes, it may not be possible to perform a cubic computation on a large network in real time on a single computer, no matter how powerful.[14]

The claim that more optimal routing is possible with a centralized approach is not true in practice. To have optimal routing, the input information must be completely accurate and up-to-date. Of course, with any realistic centralized algorithm, the input data will no longer be completely accurate when it arrives at the center. Similarly, the output data—the routing decisions—will not go into effect at the nodes until some time after they have been determined at the center.

Distributed routing algorithms, whether fixed random, fixed alternate, or adaptive, may contain temporary loops, that is, a packet may traverse a complete circle while the algorithm adapts (or simulates adaptation in fixed strategies) to network change. Proponents of centralized routing often argue that such loops can best be avoided by centralization of the computation. However, because of the time lags cited above, there may indeed be loops during the time of propagation of a routing update when some nodes have adopted the new routes and other nodes have not.

A centralized routing algorithm has several inherent weaknesses in the updating procedure, the first being unreliability. If the RCC should fail, or the node to which it is connected goes down, or the lines around that node fail, or a set of lines and nodes in the network fail so as to partition the network into isolated components, then some or all of the nodes in the network are without any routing information. Of course, several steps can be taken to improve on the simple centralized policy. First, the RCC can have a backup computer, either doing another task until a RCC failure, or else on hot standby. This is not sufficient to meet the problem of network failures, only local outages, but it is necessary if the RCC computer has any appreciable failure rate. Second, there can be multiple RCCs in different locations throughout the network, and again the extra computers can be in passive or active standby. Here there is the problem of identifying which center is in control of which nodes, since the nodes must know to which center to send their routing input data.

* A much more detailed discussion is given in Reference 14.

A related difficulty with centralized algorithms lies in the fact that when a node or line fails in the network, the failed component may have been on the previously best path between the RCC and the nodes trying to report the failure. In this case, just at the time the RCC needs routes over which to receive and transmit routing information, no routes are available; the availability of new routes requires the very change the RCC is unsuccessfully attempting to distribute. Solutions which have been proposed to solve this "deadlock" are slow, complicated, awkward, and frequently rely on the temporary use of distributed algorithms.[17]

Finally, centralized algorithms can place heavy and uneven demands on network line bandwidth; near the RCC there is a concentration of routing information going to and from the RCC. This heavy line utilization near the center means that centralized algorithms do not grow gracefully with the size of the network and, indeed, this may place an upper limit on the size of the network.

Isolated adaptive algorithms

One of the primary characteristics of an isolated algorithm which attempts to adapt to changing conditions is that it takes on the character of a heuristic process: it must "learn" and "forget" various facts about the network environment. While such an approach may have an intuitive appeal, it can be shown rather simply that heuristic routing procedures are unstable and are therefore not of interest for most practical network applications. The fundamental problem with isolated adaptive algorithms is that they must rely on indirect information about network conditions, since each node operates independently and without direct knowledge of or communication with the other nodes.

There are two basic approaches to be employed, separately or in tandem, to the process of learning and forgetting. We call these approaches positive feedback and negative feedback. One way to implement positive feedback was suggested by Baran as part of his hot-potato routing doctrine.[18] Each node increments the handover number in a packet as it forwards the packet. Then the handover number is used in a "backwards learning" technique to estimate the transit time from the current node to the source of the packet. Clearly, this scheme has drawbacks because it lacks any *direct* way of adapting to changes. If no packets from a given source are routed through a node by the rest of the network, the node has no information about which route to choose in sending a message to that source. In general, as part of a positive feedback loop, the routing algorithm must periodically try routes other than the current best ones, since it has no direct way of knowing if better routes exist. Thus, there must always be some level of traffic traveling on any route that the nodes are to learn about, since it is only by feedback from traffic that they can learn.

The other half of an adaptive isolated algorithm is the negative feedback cycle. One technique to use here is to penalize the choice of a given path when a packet is detected to have returned over the same path without being delivered to its destination. The relation of this technique to the exploratory nature of positive feedback is evident.

An adaptive isolated algorithm, therefore, has this fundamental weakness: in the attempt to adapt heuristically, it must oscillate, trying first one path and then another, even under stable network conditions. This oscillation violates one of the important goals of any routing algorithm, stability, and it leads to poor utilization of network resources and slow response to changing conditions. Incorrect routing of the packets during oscillation increases delay and reduces effective throughput correspondingly. There is no solution to the problem of oscillation in such algorithms. If the oscillation is damped to be slow, then the routing will not adapt quickly to improvements and will therefore declare nodes unreachable when they are not, with the result that suboptimal paths will be used for extended periods. If the oscillation is fast, then suboptimal paths will also be used much of the time, since the network will be chronically full of traffic going the wrong way.

Distributed adaptive algorithms

In our experience, distributed adaptive algorithms have none of the inherent limitations of the above algorithms; e.g., not the inherent unreliability and inefficiency of non-adaptive algorithms, nor the unreliability and size limitations of centralized algorithms, nor the inherent inefficiency and instability of isolated algorithms. For example, the distributed adaptive routing algorithm in the ARPA Network has operated for five years with little difficulty and good performance. However, distributed algorithms do have some practical difficulties which must be overcome in order to obtain good performance.

Consider the following example of a distributed adaptive algorithm. Each node estimates the "distance" it expects a packet to have to traverse to reach each possible destination over each of its output lines. Periodically, it selects the minimum distance estimate for each destination and passes these estimates to its immediate neighbors. Each node then constructs its own routing table by combining its neighbors' estimates with its own estimates of distance to each neighbor. For each destination, the table is then made to specify that selected output line for which the sum of the estimated distance to the neighbor plus the neighbor's distance estimate to the destination is smallest.

Such an algorithm can be made to measure distance in hops (i.e., lines which must be traversed), delay, or any of a number of other metrics including excess bandwidth and reliability (of course, for the latter two, one must maximize rather than minimize). The above algorithm is representative of a class of distributed adaptive algorithms which we consider briefly in the remainder of this section. For simplicity of discussion we will assume that distance is measured in hops.

The first point is that distributed algorithms are slow in adapting to some kinds of change; in particular, the algorithm reacts quickly to good news, and slowly to bad news. If the number of hops to a given node decreases, the nodes soon all agree on the new, lower, number. If the hop count increases, the nodes will not believe the reports of higher counts while they still have neighbors with the old, lower values. This is demonstrated in Reference 14. Another point is that there is no way for a node to know ahead of time what the next-best or fall-back path will be in the event of a failure, or indeed if one exists. In fact, there must be some finite time, the network response time, between when a change in the network occurs and when the routing algorithm adapts to the change. This time depends on the size and shape of the network.

We have come to conclude that the routing algorithm should continue to use the best route to a given destination, both for updating and forwarding, for some time period after it gets worse. That is, the algorithm should report to the adjacent nodes the current value of the previous best route and use it for routing packets for a given time interval. We call this *hold down.*[14] One way to look at this is to distinguish between changes in the network topology and traffic that necessitate changing the choice of the best route, and those changes which merely affect the characteristics of the route, like hop count, delay, and throughput. In the case when the identity of the path remains the same, the mechanism of hold down provides an instantaneous adaptation to the changes in the characteristics of the path; certainly, this is optimal. When the identity of the path must change, the time to adapt is equal to the absolute minimum of one network response time, while the other nodes have a chance to react to the worsening of the best path and to decide on the next best path. This is optimal for any algorithm within the practical limits of propagation times.*

The routing algorithm is extremely important to network reliability, since if it malfunctions the network is useless. Further, a distributed routing algorithm has the property that all the nodes must be performing the routing computation correctly for the algorithm to be reliable. A local failure can have global consequences; e.g., one node announcing that it is the best path to all nodes. Routing messages between nodes must have checksums and must be discarded if a checksum error is detected. All routing *programs* must be checksummed before every execution to verify that the code about to be run is correct. The checksum of the program should include the preliminary checksum computation itself, the routing program, any constants referenced, and anything else which could affect its successful execution. Any time a checksum error is detected in a node, the node should immediately be stopped from participating in the routing computation until it is restored to correct operation again.

Node-to-node transmission procedures

In this section we discuss some of the issues in designing node-to-node transmission procedures, that is, the packet processing algorithms. We touch on these points only briefly since many of them are simple or have been discussed previously. Note that many of these issues occur again in the discussion of source-to-destination transmission procedures.

Buffering and pipelining

As we noted in discussing memory requirements, the amount of node-to-node packet buffering needs to equal the product of the circuit rate times the expected acknowledgment delay in order to get full line utilization. It may also be efficient to provide a small amount of additional buffering to deal with statistical fluctuations in the arrival rates, i.e., to provide queueing. These requirements imply that the nodes must do bookkeeping about multiple packets, which raises the several issues discussed next.

Error control

We have discussed many of the aspects of node-to-node error control above: the need for a packet checksum, its size, the basis of the acknowledgment/retransmission system, the decision on whether the line is usable, and so on. These procedures are critical for network reliability, and they should therefore run smoothly in the face of any kind of node or circuit failure. Where possible, the procedures should be self-synchronizing; at least they should be free from deadlock and easy to resynchronize.[10]

Storage allocation and flow control

Storage allocation can be fairly simple for the packet processing algorithms. The sender must hold a copy of the packet until it receives an acknowledgment; the receiver can accept the packet if it is without error and there is an available buffer. The receiver should not use the last free buffer in memory, since that would cut off the flow of control information such as routing and acknowledgments. In accepting too many packets, there is also the chance of a storage-based deadlock in which two nodes are trying to send to each other and have no more room to accept packets. This is explained fully in Reference 20.

The above implies that the flow control procedures can also be fairly simple. The need to buffer a circuit can be expressed in a quantitative limit of a certain number of packets. Therefore, the node can apply a cut-off test per line as its flow control throttle. More stringent rules can be used, but may be unnecessary.

Priority

The issue of priority in packet processing is quite important for network performance. First of all, the concept

* This is a very simplified description of hold down. A more complete description states in detail when hold down should be invoked and for what duration. Such a description may be found in Reference 14 and more is being learned.[19]

of two or more priority levels for packets is useful in decreasing queueing delay for important traffic. Beyond this, however, careful attention must be paid to other kinds of transmissions. Routing messages should go with the highest priority, followed by acknowledgments (which can also be piggybacked in packets). Packet retransmissions must be sent with the next highest priority, higher than that for first transmission of packets. If this priority is not observed, retransmissions can be locked out indefinitely. The question of preemptive priority (i.e., stopping a packet in mid-transmission to start a higher priority one) is one of a direct tradeoff of bandwidth against delay since circuit bandwidth is wasted by each preemption.

Packet size

There has been much thought given in the packet-switching community to the proper size for packets. Large packets have a lower probability of successful transmission over an error-prone telephone line (and this drives the packet size down), while overhead considerations (longer packets have a lower percentage overhead) drive packet size up. The delay-lowering effects of pipelining become more pronounced as packet size decreases, generally improving store-and-forward delay characteristics; further, decreasing packet size reduces the delay that priority packets see because they are waiting behind full length packets. However, as the packet size goes down, effective throughput also goes down due to overhead. Metcalfe has previously commented on some of these points.[21]

Kleinrock and Naylor[22] recently suggested that the ARPA Network packet size was suboptimal and should perhaps be reduced from about 1000 bits to 250 bits. This was based on optimization of node buffer utilization for the observed traffic mix in the network. However, in Reference 23, we point out that the relative cost of node buffer storage vs. circuits is possibly such that one should not try to optimize node buffer storage. The true trade-off which governs packet size might well be efficient use of phone line bandwidth (driving packet size larger) vs. delay characteristics (driving packet size smaller). If buffer storage is limiting, perhaps one should just buy more. Further, it is probably true that if one is trying for high bandwidth utilization, buffer size must be large. That is, high bandwidth utilization probably implies the use of large packets, which implies full buffers; when idle, the buffer size does not matter.

As noted above, the choice of packet is influenced by many factors. Since some of the factors are inherently in conflict, an optimum is difficult to define, much less find. The current ARPA Network packet size of about 1000 bits is a good compromise. Other packet sizes (e.g., the 2000 bits used in several other networks) may also be acceptable compromises. However, note that a 2000-bit packet size generally means a factor of two increase in delay over a 1000-bit packet size, because even high priority short packets will be delayed behind normal long packets which are in transmission at each node. The use of preemptive priority might make longer packet sizes efficient.

Davies and Barber[24] are often quoted as recommending a minimum length "packet" of about 2000 bits because they have concluded that most of the messages currently exchanged within banks and airlines fit nicely in one packet of this size. To clarify this point, we note that they use the term "packet" for the unit of information we call a "message" and thus are not actually addressing the issue of packet size. We discuss message size below.

SOURCE-TO-DESTINATION SOFTWARE DESIGN

In this section we discuss the end-to-end transmission procedures and the division of responsibility between the Hosts and nodes.

End-to-end transmission procedures

There is a considerable controversy at the present time over whether or not a store-and-forward subnetwork of nodes should concern itself with end-to-end transmission procedures. Many workers[2] feel that the subnetwork should be close to a pure packet carrier with little concern for maintaining message order, for high levels of correct message delivery, for message buffering in the subnetwork, etc. Other workers, including ourselves,[23] feel that the subnetwork should take responsibility for many of the end-to-end message processing procedures. Of course, there are some workers who hold to positions in between.[3] However, many design issues remain constant whether these functions are performed at Host level or subnetwork level, and we discuss these constants in this section.

Buffering and pipelining

As noted earlier in this paper, any practical network must allow multiple messages simultaneously in transit between the source and the destination, to achieve high throughput. If, for example, one message of 2000 bits is allowed to be outstanding between the source and destination at a time, and the normal network transit for the message including destination-to-source acknowledgment is 100 milliseconds, then the throughput rate that can be sustained is 20,000 bits per second. If slow lines, slow responsiveness of the destination Host, great distance, etc., cause the normal network transit time to be half a second, then the throughput rate is reduced to only 4,000 bits per second. Likewise, we think that pipelining is essential for most networks to improve delay characteristics; data should travel in reasonably short packets.

To summarize, low delay requirements drive packet size smaller, network and Host lines faster, and network paths shorter (i.e., fewer node-to-node hops). High throughput requirements drive the number of packets in flight up, packet overhead down, and the number of alternative paths up.

Error control

We consider source-to-destination error control to comprise three tasks: detecting bit errors in the delivered messages, detecting missing messages or pieces of messages, and detecting duplicate messages or pieces of messages.

The former task is done in a straightforward manner through the use of checksums. A checksum is appended to the message at the source and the checksum is checked at the destination; when the checksum does not check at the destination, the incorrect message is discarded, requiring it to be retransmitted from the source. Several points about the manner in which checksumming should be done are worthy of note: (a) If possible, the checksum should check the correctness of the resequencing of the messages which possibly got out of order in their traversal of the network. (b) A powerful checksum is more efficient than alternative methods such as replication of a critical control field; it is better to extend the checksum by the number of bits that would have been used in the redundant field. (c) Unless encryption is desirable for some other reason it is simpler (and just as safe) to prevent delivery of a message to an incorrect Host through the use of a powerful checksum than it is to use an encryption mechanism. (d) Node-to-node checksums do not fulfill the same function as end-to-end checksums because they check only the lines, not the nodes.

An inherent characteristic of packet-switching networks is that some messages or portions of messages (i.e., packets) will fail to be delivered, and there will be some duplicate delivery of messages or portions of messages, as described in the section on network properties.*

Missing messages can be detected at the destination through the use of one state bit for each unit of information which can be simultaneously traversing the network. An interesting detail is that for the purposes of missing message detection, the state bits used must precisely cycle through all possible states. For example, stamping messages with a time stamp does nothing for the process of missing message detection because, unless a message is sent for every "tick" of the time stamp, there is no way to distinguish the case of a missing message from the case where no messages were sent for a time.

Duplicate messages can be detected with an identifying sequence number such that messages which arrive from a prior point in the sequence are recognized as duplicates. What should be noted carefully here is that duplicate messages can arrive at the destination up to some time, possibly quite long, after the original copy, and the sequence number must not complete a full cycle during this period. For example, if a network goal is to be able to transmit 200 minimum length messages per second from the source to the destination and each needs a unique sequence number, and if it is possible for messages to arrive at the destination up to 15 seconds after initial transmission from the source, then the sequence number must be able to uniquely identify at least 3000 packets. It is usually no trouble to calculate the maximum number of messages that can be sent during some time interval. What is more difficult is to limit the maximum time after which duplicate messages will no longer arrive at the destination. One method is to put a timer in each message which is counted down as the message traverses the network; if the timer ever counts out, the message is discarded as too old, thus guaranteeing that no messages older than the initial setting of the timer will be delivered to the destination. Alternatively, one can calculate approximately the maximum arrival time through study of all the worst case paths through the network and all the worst case combinations of events which might cause messages to loop around in the network for excessive lengths of time; this seems to work reasonably well in practice.

In either case, there certainly must be mechanisms to resynchronize the sequence numbers between the source and the destination at node start-up time, to recover from a node failure, etc. A good practice is to resynchronize the sequence numbers occasionally even though they are not known to be out of step. A good frequency with which to do redundant resynchronization would be every time a message has not been sent for longer than the maximum delivery time. In fact, this is the maximum frequency with which the resynchronization can be done (without additional mechanisms); if duplicates are to be detected reliably, the sequence number at the destination *must* function without disruption for the maximum delivery time after the "last message" has been sent. If it is desirable or necessary to resynchronize the sequence numbers more often than the maximum time, an additional "use" number must be attached to the sequence number to uniquely identify which "instance" of this set of sequence numbers is in effect; and, of course, the packets must also carry the use number. This point is addressed in greater detail in References 25 and 26.

The next point to make about end-to-end error control is that any message going from source to destination can potentially be missing or duplicated; i.e., not only data messages but control messages. In fact, the very messages used in error control (e.g., sequence number resynchronization messages) can themselves be missing or duplicated, and a proper end-to-end protocol must handle these cases.

Finally, there must be some inquiry-response system from the source to the destination to complete the process of detecting lost messages. When the proper reply or acknowledgment has not been received for too long, the source may inquire whether the destination has received the message in question. Alternatively, the source may simply retransmit the message in question. In any case, this source inquiry and retransmission system must also function in the face of duplicated or lost inquiries and inquiry response control messages. As with the inter-node acknowledgment and retransmission system, the end-to-end acknowledgment and retransmission system must depend on positive acknowledgments from the destination to the

* Throughout the remainder of this subsection we use the word "message" to mean either messages or portions of messages (i.e., packets).

source and on explicit inquiries or retransmissions from the source. Negative acknowledgments from the destination to the source are never sufficient (because they might get lost) and are only useful for increased efficiency.

Storage allocation and flow control

One of the fundamental rules of communications systems is that the source cannot simply send data to the destination without some mechanism for guaranteeing storage for that data. In very primitive systems one can guarantee a rate of disposal of data, as to a line printer, and not exceed that rate at the data source. In more sophisticated systems there seem to be only two alternatives. Either one can explicitly reserve space at the destination for a known amount of data in advance of its transmission, or one can declare the transmitted copy of the data expendable, sending additional copies from the source until there is an acknowledgment from the destination. The first alternative is the high bandwidth solution: when there is no space, only tiny messages travel back and forth between the source and destination for the purpose of reserving destination storage. The second alternative is the low delay solution: the text of the message propagates as fast as possible. See Reference 10 for a more lengthy discussion.

In either case storage is tied up for an amount of time equal to at least the round trip time. This is a fundamental result—the minimum amount of buffering required by a communications system, either at the source or at the destination, equals the product of round trip time and the channel bandwidth. The only way to circumvent this result is to count on the destination behaving in some predictable fashion (an unrealistic assumption in the general case of autonomous communicating entities).

As we stated earlier, our experience and analysis convince us that if both low delay and high throughput are desired, then there must be mechanisms to handle each, since high throughput and low delay are conflicting goals. This is true, in particular, for the storage allocation mechanism. In several networks, e.g.,[2] mainly for the sake of simplicity, only the low delay solution has been proposed or implemented; that is, messages are transmitted from the source without reservation of space at the destination. Those people making the choice never to reserve space at the destination frequently assert that high bandwidth will still be possible through use of a mechanism whereby the source sends messages toward the destination, notes the arrival of acknowledgments from the destination, uses these acknowledgments to estimate the destination reception rate, and adjusts its transmissions to match that rate. We feel that such schemes may be quite difficult to parameterize for efficient control and therefore may result in reduced effective bandwidth and increased effective delay. If, in addition to possible discards at the destination, the network solves its internal problems by discarding packets, or if the destination Host too often solves its internal problems by discarding packets, performance will suffer further. As reported in Reference 20, contention for destination storage, which must be resolved through the discard of packets in the absence of a storage allocation mechanism, happens practically continuously under even modest traffic loads, and in a way uncoordinated with the rates and strategies of the various sources. As a result, well-behaved Hosts may unavoidably be penalized for the actions of poorly-behaved Hosts.

In addition to space to hold all data, there must also be space to hold all control messages. In particular, there must be space to record what needs to be sent and what has been sent. If a message will result in a response, there must be space to hold the response; and once a response has been sent, the information about what kind of answer was sent must be kept for as long as retransmission of that response may be necessary.

Precedence and preemption

The first point to note about precedence and preemption is that the total transit time being specified for most packet-switching networks of which we are aware is on the order of less than a few seconds (often only a fraction of a second). Thus, the traditional specifications (for example, low priority traffic must be able to preempt all other traffic so that it can traverse the network in under two minutes) no longer make much sense. When all messages traverse the network in less than a few seconds, there is generally no need to specify that top priority traffic must preempt other traffic, nor to specify the relative precedences between the other types of traffic.

Though priority is not strictly necessary for speed, it may be useful for contention resolution. It appears to us that there are three precedence and preemption strategies that are reasonable to consider for a packet-switching network. Strategy 1 is to permanently assign the resources necessary to handle high priority traffic; this guarantees the delivery time for the high priority traffic but is expensive and should only be done for limited high priority traffic. Strategy 2 is to preempt resources as necessary for high priority traffic. This can have two effects. Preempting packet buffers results in data loss; preempting internal node tables (e.g., the tables associated with packet sequence numbering) results in state information loss. State information loss means that data errors are possible which may go unreported. Strategy 3 is not to preempt resources, and to rely on the standard mechanisms with a priority ordering. This is simple for the nodes, but it does not itself guarantee delivery within a certain time.

We think the correct strategy is probably a mixture of the strategies above. Possibly some resources, on a very limited basis, should be reserved for the tiny amount of flash traffic. This guarantees minimum delay without any queueing latency. For the rest of the traffic, the normal delivery times are probably acceptable. The presence of higher priority traffic can cause gradual throttling of lower priority traffic, without loss of state information. As the time to do this graceful throttling is normally only a fraction of a second, the higher priority traffic has no real reason to demand instantaneous, information-losing preemption of the lower priority traffic.

Message size

The question is often asked: "If one increases packet size, and decreases message size until the two become the same, will not the difficult message reassembly problem be removed?" The answer is that, perhaps unfortunately, message size and packet size are almost unrelated to reassembly.

We have already noted the relationship between delay and packet size. Delay for a small priority message is, to first order, proportional to the packet size of the other traffic in the network. Thus, small packets are desirable. Larger packets become desirable only when lines become so long or fast that propagation delay is larger than transmission time.

Message size needs to be large because the overhead on messages is significant. It is inefficient for the nodes to have to address too many messages and it may be inefficient for Hosts to have too many message interrupts. The upper limit on message size is what can conveniently be reassembled, given node storage and networks delays.

When a channel has an appreciable delay, it is necessary to buffer several pieces of data in the channel at one time in order to obtain full utilization of the channel. It makes little difference whether these pieces are called packets which must be reassembled or messages which must be delivered in order.

We do not feel that the choice between single- and multi-packet messages is as important as all the controversy on the subject would lead one to believe. There is agreement that buffering many data units in transit through the network simultaneously is a necessity. Having multi-packet messages is probably more efficient (as the extra level of heirarchy allows overhead functions to be applied at the correct, i.e., most efficient, level); having single-packet messages probably offers the opportunity for finer grained storage allocation and flow control mechanisms.

Division of responsibility between subnetwork and Host

In the previous section we discussed a number of issues of end-to-end procedure design which must be considered wherever the procedures are implemented, whether in the subnetwork or in the Hosts. In this section we discuss the proper division of responsibility between the subnetwork and the Hosts.

Extent of message processing in the subnetwork

There has been considerable discussion in the packet-switching community about the amount and kind of message processing that should be done in communications subnetworks. An important part of the ARPA Network design which has become controversial is the ARPA Network system of messages and packets within the subnetwork, ordering of messages, guaranteed message delivery, and so on. In particular, the idea has been put forth that such functions should reside at Host level rather than subnetwork level.[2,27,28]

We summarize the principles usually given for eliminating message processing from the communications subnetwork: (a) for complete reliability, Hosts must do the same jobs, and therefore the nodes should not; (b) Host/Host performance may be degraded by the nodes doing these jobs; (c) network interconnections may be impeded by the nodes doing message processing; (d) lockups can happen in subnetwork message processing; (e) the node would become simpler and have more buffering capacity if it did not have to do message processing.

The last point is true although the extent of simplification and the additional buffering is probably not significant, but we believe the other statements are subject to some question. We have previously[23,25] given our detailed reasons for this belief. Here we simply summarize our main contentions about the place of message processing facilities in networks:

a. A layering of functions, a hierarchy of control, is essential in a complex network environment. For efficiency, nodes must control subnetwork resources, and Hosts must control Host resources. For reliability, the basic subnetwork environment must be under the effective control of the node program—Hosts should not be able to affect the usefulness of the network to other Hosts. For maintainability, the fundamental message processing program should be node software, which can be changed under central control and much more simply than all Host programs. For debugging, a hierarchy of procedures is essential, since otherwise the solution of any network difficulty will require investigating *all* programs (including Host programs) for possible involvement in the trouble.

b. The nature of the problem of message processing does not change if it is moved out of the network and into the Hosts; the Hosts would then have this very difficult job even if they do not want it.

c. Moving this task into the Hosts does not alleviate any network problems such as congestion, Host interference, or suboptimal performance but, in fact, makes them worse since the Hosts cannot control the use of node resources such as buffering, CPU bandwidth, and line bandwidth.

d. It is basically cheaper to do message processing in the nodes than in the Hosts and it has very few detrimental effects.

Peripheral processor connections

In a number of cases, an organization has desired to connect a large Host to a network by inserting an additional minicomputer between the main Host and the node. The general notion has been to locate the Host-Host transmission procedures in this additional machine, thus relieving the main Host from coping with these tasks. Stated reasons for this notion include:

- It is difficult to change the monitor in the main Host, and new monitor releases by the Host manufacturer pose continuing compatibility problems.
- Core or timing limitations exist in the main Host.
- It is desirable to use I/O arrangements that may already exist or be available between the main Host and the additional mini (and between the mini and the node) to avoid design or procurement of new I/O gear for the main Host.

While this approach may sound good in principle, and, in fact, may be the only possible approach in some instances, it often leads to problems.

First, the I/O arrangements between the main Host and any preexisting peripheral processor were not designed for network connection and usually present timing and bandwidth constraints that greatly degrade performance. More seriously, the logical protocols that may have preexisted will almost certainly preclude the main Host from acting as a general purpose Host on the network. For instance, while initial requirements may only indicate a need for simple file transfers to a single distant point, requirements tend to change in the face of new facilities, and the network cannot then be used to full advantage.[29]

Second, the peripheral processor and its software are often provided by an outside group, and the Host organization may know even less about their innards than they know about the main Host. The node is centrally maintained, improved, modified, and controlled by the Network Manager, but the peripheral processor, while an equally foreign body, is not so fortunate. This issue alone is crucial; functions that do not belong in the main Hosts belong in centrally monitored network equipment. Note that it is exactly those Host groups who are unwilling to touch the main Host's monitor who will be unlikely to be able to make subtle improvements in the protocols, error message handling, and timing of the peripheral processor. From a broader economic view, common functions belong in the network and should be designed once; the peripheral processor approach is a succession of costly special cases and the total cost is greatly escalated.

The long term solution to the dilemma is to have the various manufacturers support hardware and software interfaces that connect to widely used networks. This is not likely to occur until commercial networks exist and are widely available. In the meantime, potential Host organizations that wish to use early networks (like the ARPA Network) should try to find ways to put the network connection directly into the main Host. An anthropomorphic illustration may be helpful: the network is, among other things, a set of standardized protocols or languages. A potential network Host is in the position of a person who needs to have dealings with people who speak a language he does not know. If he does not want to learn the language, he can indeed opt for using an interpreter, but performance is poor, the process is very inconvenient, expensive, and unpleasant, and subtle meaning is always lost. The situation is quite similar when a Host tries to work through a peripheral processor. If a Host wishes to interact with a network, it is usually unrealistic to try to make the Host think that the network is a card reader or some other familiar peripheral. As usual, you get what you pay for.

Other message services

One commonly suggested design requirement is for storage in the communications subnetwork, usually for messages which are currently undeliverable because a Host or a line is down. This requirement should have no effect whatsoever on the design of the communications part of the network; it is an orthogonal requirement which should be implemented by providing special storage Hosts at strategic locations in the network. These can be at every node, at a few nodes, or at a single node, depending on the relative importance of reliability, efficient line utilization, and cost.

Another commonly suggested design requirement is for the communications subnetwork to provide a message broadcast capability; i.e., a Host gives a message to its node along with a list of Host addresses and the nodes somehow send copies to all the Hosts in the list. Again we believe that such a requirement should have no effect on the design of the communications part of the network and that messages to be broadcast should be sent to a special Host (perhaps one of the ones in the previous paragraph) for such broadcast.

CONCLUSION

There has now been considerable experience with the design of packet-switching networks and several groups (ours included) believe that they have come to understand many of the fundamental design issues. On the other hand, packet switching is still in its youth, and there are many new issues to be explored. Such new issues include, among others: (a) the techniques for transferring packet-switching technology from its initial limited R&D implementations to widespread production implementations; (b) the methods whereby the newly available packet-by-satellite technology can be utilized in packet-switching networks; (c) transmission of speech through packet-switching networks; (d) packet transmission by radio; (e) interconnection of packet-switching networks; and (f) effects of packet-switching networks on Host operating system design. Several other papers in these same proceedings cover in detail some of the new design issues just mentioned[30,31,32,33] and we plan to address some of these new issues ourselves in the near future.

ACKNOWLEDGMENTS

Since 1969 our research on packet-switching network design has been encouraged and supported by the Information Processing Techniques office of the Advanced Research Projects Agency. Many of our colleagues at Bolt Beranek and Newman Inc. have participated in our research. Through the ARPA Network and through the International Network Working Group we have been fortunate to be able to receive frequently incisive critiques of our work and to have the opportunity to study the work of others. In particular, over the past year our interactions with Holger Opderbeck and his colleagues at UCLA have been enlightening. Finally, we acknowledge the substantial assistance of Robert Brooks and Barbara Erwin with the preparation of the manuscript, and we acknowledge the very useful comments of Drs. J. Burchfiel, W. Hawyrlko, R. Metcalfe, and J. Postel, who reviewed the presentation and content of this paper.

REFERENCES

1. Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," *AFIPS Conference Proceedings*, Vol. 36, June 1970, pp. 551-567; also in *Advances in Computer Communications*, W. W. Chu (ed.), Artech House Inc., 1974, pp. 300-316.
2. Pouzin, L., "Presentation and Major Design Aspects of the Cyclades Computer Network," *Proceedings of the Third ACM Data Communications Symposium*, November 1973, pp. 80-88.
3. Brant, G. J. and G. J. Chretien, "Methods to Control and Operate a Message-Switching Network," *Computer-Communications Network and Teletraffic*, Polytechnic Press of the Polytechnical Institute of Brooklyn, Brooklyn, N.Y., 1972.
4. Barber, D. L. A. (ed.), *A Specification for a European Informatics Network*, Co-Opèration Europèenne dans le Domaine de la Recherche Scientifique et Technique, January 4, 1974.
5. Rosner, R. D., "A Digital Data Network Concept for the Defense Communications System," *Proceedings of the National Telecommunications Conference*, Atlanta, November 1973, pp. 22C1-6.
6. Davies, D. W., K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A Digital Communication Network for Computers Giving Rapid Response at Remote Terminals," *Proceedings of the ACM Symposium on Operating Systems Principles*, October 1967.
7. Auerbach Publishers Inc., *Public Packet Switching Networks*, Data Processing Manual No. 3-08-04, 1974.
8. Despres, R. F., "A Packet Switching Network with Graceful Saturated Operation," *Proceedings of the First International Conference on Computer Communication*, October 1972, pp. 345-351.
9. Pouzin, L., *Basic Elements of a Network Data Link Control Procedure* (*NDLC*), INWG 54, NIC 30375, January 1974, a limited number of copies available for the cost of reproduction and handling from INWG, c/o Prof. V. Cerf, Digital Systems Laboratory, Stanford, CA. 94305.
10. McQuillan, J. M., W. R. Crowther, B. P. Cosell, D. C. Walden, and F. E. Heart, "Improvements in the Design and Performance of the ARPA Network," *AFIPS Conference Proceedings*, Vol. 41, December 1972, pp. 741-754.
11. Heart, F. E., S. M. Ornstein, W. R. Crowther, and W. B. Barker, "A New Minicomputer/Multiprocessor for the ARPA Network," *AFIPS Conference Proceedings*, Vol. 42, June 1973, pp. 592-537; also in *Selected Papers: International Advanced Study Institute, Computer Communication Networks*, R. L. Grimsdale and F. F. Kuo (eds.), University of Sussex, Brighton, England, September 1973; also in *Advances in Computer Communications*, W. W. Chu (ed.), Artech House Inc., 1974, pp. 329-337.
12. Ornstein, S. M., W. R. Crowther, M. F. Kraley, R. D. Bressler, A. Michel, and F. E. Heart, "Pluribus—A Reliable Multiprocessor," these proceedings.
13. Frank, H., R. E. Kahn, and L. Kleinrock, "Computer Communications Network Design—Experience with Theory and Practice," *AFIPS Conference Proceedings*, Vol. 40, June 1972, pp. 255-270; also in *Networks*, Vol. 2, No. 2, 1972, pp. 135-166; also in *Advances in Computer Communication*, W. W. Chu (ed.), Artech House Inc., 1974, pp. 254-269.
14. McQuillan, J. M., *Adaptive Routing Algorithms for Distributed Computer Networks*, BBN Report No. 2831, May 1974, available from the National Technical Information Service, AD781467.
15. Grange, J. L., *Cyclades Network*, personal communication.
16. Floyd, R. W., "Algorithm 97, Shortest Path," *CACM* 5 (6), June 1962, p. 345.
17. Gerla, M., "Deterministic and Adaptive Routing Policies in Packet-Switched Computer Networks, *Proceedings of the Third ACM Data Communications Symposium*, November 1973, pp. 23-28.
18. Baran, P., *On Distributed Communications: I. Introduction to Distributed Communications Networks*, Rand Corp. Memo RM-3420-PR, August 1964, p. 37.
19. Opderbeck, H. and W. Naylor, *ARPA Network Measurement Center*, personal communication.
20. Kahn, R. E. and W. R. Crowther, "Flow Control in a Resource-Sharing Computer Network," *Proceedings of the Second ACM/IEEE Symposium on Problems in the Optimization of Data Communications Systems*, Palo Alto, California, October 1971, pp. 108-116; also in *IEEE Transactions on Communications*, Vol. COM-20, No. 3, Part II, June 1972, pp. 539-546.
21. Metcalfe, R. M., *Packet Communication*, Massachusetts Institute of Technology Project MAC Report MAC TR-114, December 1973.
22. Kleinrock, L. and W. Naylor, "On Measured Behavior of the ARPA Network," *AFIPS Conference Proceedings*, Vol. 43, May 1974, pp. 767-780.
23. Crowther, W. R., F. E. Heart, A. A. McKenzie, J. M. McQuillan, and D. C. Walden, *Network Design Issues*, BBN Report No. 2918, November 1974, to be available from National Technical Information Service.
24. Davies, D. W., and D. L. A. Barber, *Communication Networks for Computers*, London: John Wiley and Sons, 1973.
25. McQuillan, J. M., "The Evolution of Message Processing Techniques in the ARPA Network," to appear in *International Computer State of the Art Report No. 24: Network Systems and Software*, Infotech, Maidenhead, England.
26. Tomlinson, R. S., *Selecting Sequence Numbers*, INWG—Protocol Note #2, August 1974, available as with Reference 9.
27. Cerf, V. and R. Kahn, "A Protocol for Packet Network Intercommunications," *IEEE Transactions on Communications*, Vol. COM-22, No. 5, May 1974, pp. 637-648.
28. Cerf, V., *An Assessment of ARPANET Protocols*, RFC 635, NIC 30489, April 1974, available as with Reference 9.
29. Metcalfe, R. M., "Strategies for Operating Systems in Computer Networks," *Proceedings of the ACM National Conference*, August 1972, pp. 278-281.
30. Retz, D. L., "Operating System Design Considerations for the Packet Switching Environment," these proceedings.
31. Forgie, J. W., "Speech Transmission in Packet-Switched Store-and-Forward Networks," these proceedings.
32. Lam, S. S., and L. Kleinrock, "Dynamic Control Schemes for a Packet Switched Multi-Access Broadcast Channel," these proceedings.
33. Kahn, R. E., "The Organization of Computer Resources into a Packet Radio Network," these proceedings.

The organization of computer resources into a packet radio network

by ROBERT E. KAHN

Advanced Research Projects Agency
Arlington, Virginia

INTRODUCTION

In this paper, we describe the use of packet radio communication for organizing computer resources into a computer communications network. A system to demonstrate the packet radio concept is being developed by the Advanced Research Projects Agency with initial testing in the San Francisco area beginning in 1975. The attributes of this system are presented and its application to mobile radio communications and computer architecture is briefly discussed.

The development of packet switching has made possible the economic sharing of computer resources[23,24,36] over a wide geographic area and, as a valuable byproduct, it has provided an effective alternative to circuit switching in providing error-free wideband communication networks.[27,30] The basic architecture of a resource sharing computer network includes Host computers connected to one or more packet switches which may be colocated or remote from the Hosts. The packet switches are interconnected by point to point data circuits according to a topological design which results in low cost networks for a given target throughput, reliability and delay.[14,29] For a given packet switching technology, it is possible to increase network throughput greatly by assembling a higher performance switch out of a cluster of lower performance switches (see Figure 1) and by providing many more circuits between clusters.[22] An alternate approach which uses multiple minicomputers to obtain a higher performance switch is described in Reference 21.

The use of packet broadcasting techniques for interconnection becomes attractive when the number of minicomputers (or microprocessors!) is sufficiently large and the overall traffic flow is small. The use of wire "busses" for packet broadcasting appears certain to be an effective interconnection technique. However, packet radio provides another alternative that may be useful for organizing the communications among a large or even a small number of computer resources regardless of the physical setting; inside a box, within a room, or throughout a wide geographic area (see Figure 2). In addition to its utility for mobile communications, packet radio may eventually result in the development of improved techniques for maintenance, breadboarding, and packaging of computer equipment.

For a geographically distributed network, economic studies have shown that the cost of local distribution for a large user population can be a significant part of the overall system cost.[11] For this reason alone, it would be desirable to identify more economic techniques for local data distribution than the use of telephone lines. Some progress in this direction has already taken place[28] and further development of cable systems is expected. However, even if the cost of telephone access lines were not a dominant factor, an effective means of obtaining mobile access would still be required. This has provided one incentive for the development of a local radio distribution system. The burst characteristics of computer communication[31] will surely be significantly different from the characteristics of mobile radio telephone. By using packet switching over radio channels, one should be able to achieve increased utilization of the frequency spectrum for computer communications than with conventional fixed channel allocation techniques.

The progress in integrated circuit electronics, low power displays and microprocessors makes it feasible to develop a personal terminal which may be conveniently carried about by an individual.[3,8] This capability allows a user with access to a radio network to be continually in communication with people and computer resources or, selectively, to avoid any and all communications if desired. This latter capability becomes increasingly important as computers and communication capabilities are brought increasingly close to the individual.

The Aloha system

Several years ago, researchers at the University of Hawaii, noting the unusually high error rates on the local telephone lines which adversely affected their ability to remotely utilize the university computer, proposed a research program to investigate the use of burst radio transmission in place of telephone lines for error-free line of sight communications to the computer center. This led to the development of the Aloha System at the university, a set of terminals linked directly to the computer center by UHF packet radio.[7] A discussion (in retrospect) of the lessons learned in the Aloha System experience is given in a companion paper.[6]

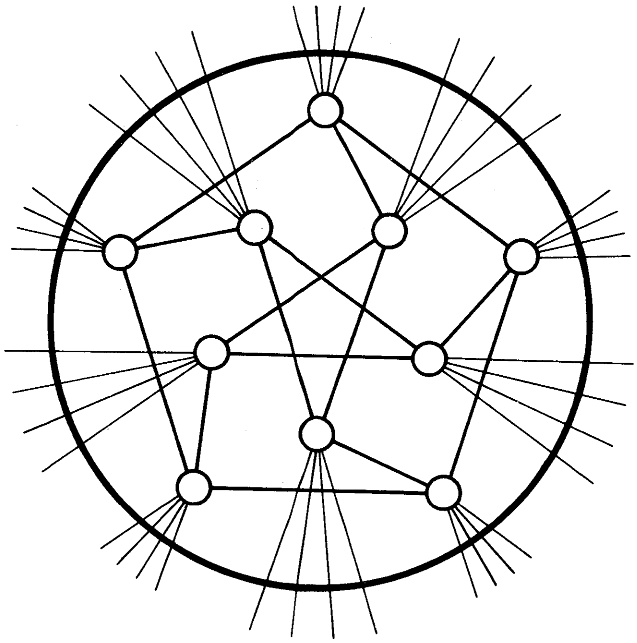


Figure 1—Packet switch formed by clustering

A simplified schematic diagram of the original Aloha system is given in Figure 3. In this system, all terminals were in line of sight of the antenna atop the computer center which served as the hub or station for all communications. Subsequently, a simple form of relay was developed to allow the range of the system to be extended over the mountains of Oahu and to the other islands.

For some applications, the Aloha architecture as developed by Hawaii represents a good choice without modification. For other applications where it is desirable to obtain coexistence with possibly different systems in the same frequency band, anti-jam protection, authentication, or direct communication by a ground radio network between users over a wide area, the Aloha system would have to be extended. In this paper, we discuss a packet radio system which is capable of meeting these added requirements and which is a network extension of the basic Hawaii work.

The main attributes of the packet radio system to be described in this paper which differ from the original Alohanet are:

(1) Distributed control of the network management functions among multiple stations for reliability, and the use of a netted array of possibly redundant repeaters for area coverage as well as reliability.
(2) The use of spread spectrum signalling for coexistence with other possibly different systems in the same band and for anti-jam protection. Surface acoustic wave technology is a viable current choice for matched filtering in the receiver.[12]
(3) The provision of authentication and anti-spoof mechanisms.
(4) The use of system protocols that include network mapping to locate and label repeaters, route determination and resource allocation, remote debugging and other distributed network functions.
(5) The use of various implementation techniques to provide efficient operational equipment such as repeater power shutdown except while processing packets.

The Alohanet has served as a useful model of a system which has a single central station with many distributed terminals within line of sight. Considerable progress has been made in analyzing this model.[25,26,32,33] In addition to random aloha and slotted aloha, the techniques analyzed have included carrier sense multiple access, busy tone multiple access and split-channel reservation multiple access.[5,15]

The packet radio system described in this paper is predicated upon the existence of an array of low cost repeaters and the need for reliable backup of all the critical system functions. It has served from the beginning as a useful model of a multiple station repeatered network. Analysis of this model has proven to be considerably more difficult than the single station model with terminals within line of sight. Consequently, simulation techniques have been extensively used for the packet radio network design.[2] We shall refer to this multi-station repeatered network as the Radionet to distinguish it from the original Alohanet at the University of Hawaii.

Frequency management

Frequency management has become a topic of considerable importance for both commercial and military use. The spectrum is already heavily crowded and efforts

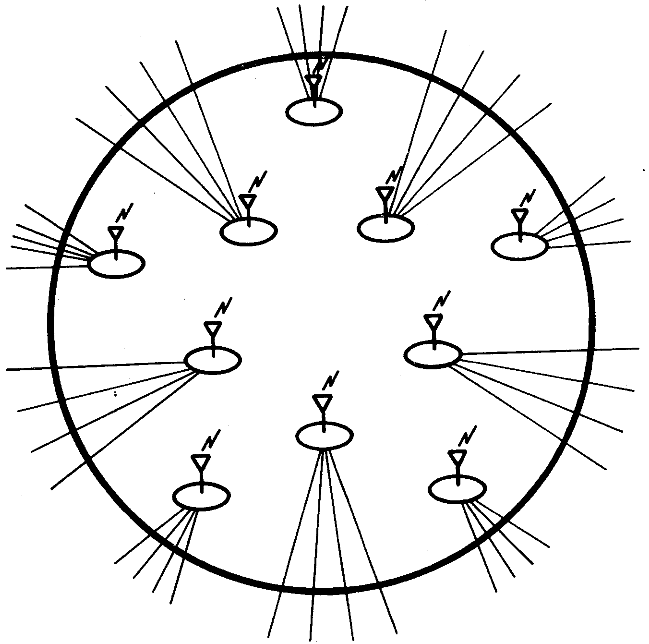


Figure 2—Packet switch using radio transmission

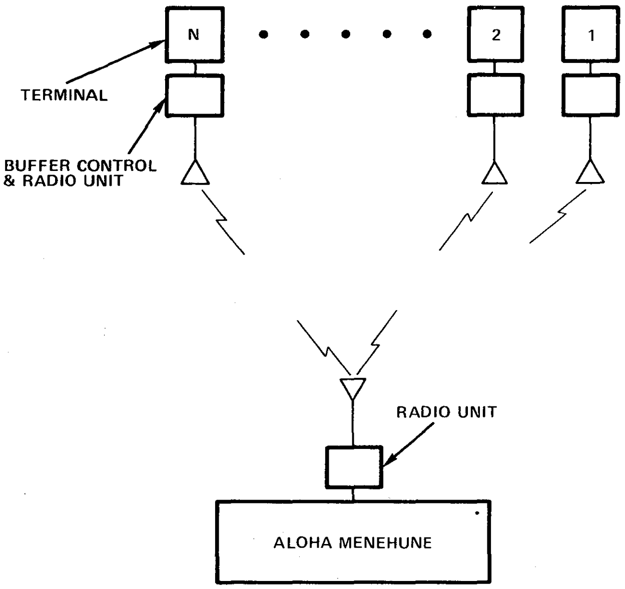


Figure 3—The original Aloha System

are constantly being made to obtain more effective use of the spectrum.[13] For example, there would seem to be no valid reason why a Radionet could not coexist in the same band as broadcast TV with virtually no adverse technical effect. Although laboratory tests of data under voice and data under video are known to have occurred, we are aware of no efforts to carry out coexistence tests in the broadcast TV band. It would not be surprising if both systems could coexist in the same band to mutual advantage.

New frequency allocations and assignments are becoming increasingly difficult to obtain due to the existence of prior assignments. This has resulted in a continual push, in the one direction, for frequency allocations at higher and higher frequencies. Non-essential use of the radio spectrum is likely to meet with pressure for movement to cable or other suitable channels. There is obviously a practical limit to the rate at which technological development can move the frequency frontiers and better techniques are needed for utilizing spectrum which is already handled by existing technology.

According to current doctrine, spectrum is assigned roughly in accordance with each users stated requirements, which are usually worst case requirements. Once allocated to a given user, the frequency band is not usually available for use by others in the same area. Joint use is presumed to cause mutual interference and thus degrade at least one and possibly both systems. This would be an effective management technique if each assigned band were actually used most of the time. In Figure 4a, we show a set of N channels each allocated to a single user. For computer communications, the utilization of each channel would typically be on the order of a few percent or less. If each channel were used approximately p percent of the time on the average, the total utilization of the assigned band would also be about p percent and could not be increased without employing a different strategy. It appears that a factor of 10-100 more utilization can be obtained for burst type traffic with the use of multiple access techniques.

On the other hand, if each of the N channels could be scheduled for transmission of one packet and then released, approximately N/p users could be served with equivalent service. Approximately the same service could also be achieved by pooling all the users into a single channel N times as wide.[35] In Figure 4b, we illustrate the spectral occupance of a common band by a set of users who coexist with each other.

In a spread spectrum multiple access system, additional bandwidth is used to provide protection against unwanted interference. However, the spread system can self interfere with itself in multiple access mode as each signal of received power P contributes P/K noise power to the output of a spread spectrum receiver, where K is the ratio of the spread bandwidth to the unspread band. Although it doesn't seem straightforward to increase the utilization by a factor of K without control of system timing to within a small fraction of 1/bandwidth, additional users may be simultaneously permitted to access the system without system wide control of timing, provided the self interference does not become too great. Thus, one pays a price for spreading the spectrum, but one also receives the advantages of a multiple access anti-jam system. Multiple access coding techniques may also be useful to increase the overall system efficiency.[16-18]

TARGET SYSTEM REQUIREMENTS

In this section the overall system objectives and the basic requirements to be met by the Radionet are discussed. The system consists of terminals and stations linked together by line of sight radio repeaters. The stations are minicomputers which provide system control; the terminals are hand-held devices, I/O consoles, computers, sensors, etc. We include Host computers in the general category of terminals. The repeaters are simple relay devices which provide network area coverage for terminals and for one or more stations.

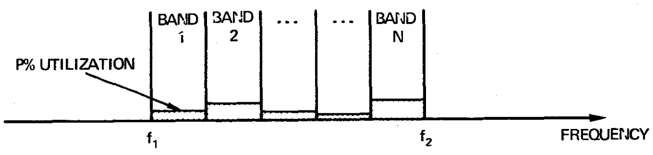


Figure 4a—Fixed frequency allocations

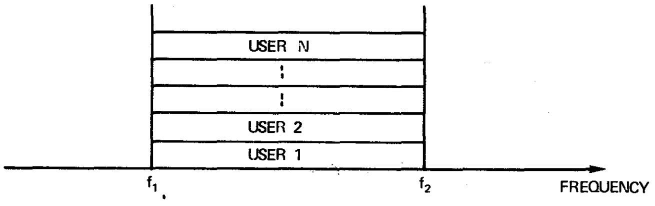


Figure 4b—Utilization of a common frequency band

One could also envision a substantially different kind of radio network where increased capability is resident in the repeater. The intent here, however, is to keep to a practical minimum all the functions that a repeater performs and to delegate the rest to the stations.

No attempt is made in this paper to compare the packet radio techniques with standard mobile radio techniques and assignment methods. The interested reader is referred to Reference 37 for background information. Comparative studies of these systems will no doubt be a topic for discussion in the coming years.

Computer communications

The system should be capable of meeting the requirements for mobile communication with computers including real-time speech communication to and from computers as well as handling data to or from portable digital terminals, various types of Host computers, etc. Since computer communication is characterized by a high ratio of peak to average traffic, a multiple access radio system is appropriate since it provides shared use of a common radio channel and is therefore expected to be more efficient than a system which dedicates resources to users with low duty cycles. A peak user data rate of 100 kilobits/second is desirable to meet the need for the rapid transfer of files and for real-time response.

The packet radio system should appear to a user as if a direct connection exists between the user and the destination. The operation of the Radionet should otherwise be transparent to the end user. For initial testing purposes the coverage area should have a diameter of at least 100 miles. The design of the radio net should be capable of extension to handle increasingly larger geographic areas with attendant increases in cost, delay, and amount of equipment.

Coexistence

The system should be able to share a common frequency band with other (possibly different) systems. The advantages of shared frequency bands are: (1) that certain equipments for different systems can be made compatible at the digital level allowing internetting to be conveniently achieved, if desired. This capability could have striking economic impact in situations where separate radio nets with separate equipment and separate frequency bands are currently established. With common equipment types and a common band, the separation could be achieved via packet labels rather than by using different bands and incompatible equipment, (2) that shared operation can result in better utilization of the frequency spectrum, (3) that the system may be introduced into a band which is currently assigned to one or more other users without first requiring the other users to vacate the band and without mutual interference, (4) that the system shall inherently be capable of providing some degree of protection against unwanted interference. Spread spectrum signalling can assist in achieving this objective and is desirable for anti-jam communications.[19]

Mobile communications

It is desirable for the packet radio technology to be able to serve users whether on land, at sea or in the air. Relative speeds of several thousands of miles/hr may occur among high speed aircraft. In the initial testing, the technology is only required to serve speeds such as might occur with ground vehicles or while walking. A target speed of one hundred miles per hour is sufficient to conduct tests with automobiles and is an appropriate initial choice. Doppler and system timing considerations are being studied to insure that higher speed vehicles can be incorporated at a later time, if desired.

No special demands should be placed on the user of the system in order that he be able to use the system while moving within its specified boundaries. In particular, the method of utilization should be the same for a user at rest and a user in motion and the performance of the system should be nominally the same regardless of the users location within the boundaries of the Radionet. Further study is required to determine what, if any, degradation results from trying to achieve exactly equal performance (as opposed to nominally equal) within some widespread portion of the Radionet.

Traffic handling

Experimental evidence[34] indicates that a maximum packet size of 1000 bits seems to be a satisfactory choice for the vast majority of computer communication requirements. For portable digital terminals (as with real-time computer speech input) packet sizes of a few hundred bits are more than ample, and character by character transmission is often needed. The use of special protocols[20] for handling character transmission will help in reducing the number of single character packets. This reduction is achieved by allowing transmission to occur only when an appropriate character (such as a command terminator) occurs or when sufficient characters have been accumulated. The Radionet should therefore be able to handle packet sizes from zero to 1000 bits and, for interactive response, an average delivery time of 0.1 seconds is desired for the longest packets within the 100 mile coverage area.

Rapid and convenient deployment

The individual elements of the packet radio system should be constructed so that they can be installed in the field with little delay. Technology has progressed to the stage where the individual elements that must be deployed can be made small enough and light enough to be carried by hand. There should be no requirement for careful alignment or other tailoring of the equipment during installation. Rather, it should be insensitive to the specific characteristic of the setting. Omni-directional antennas can be used and the system can be made to learn of the presence or absence of individual components automatically so as to avoid the need for careful coordination among different sites in the deployment or installation process.

An omni-directional radiation pattern is also desirable for area coverage of mobile terminals. In specific cases, however, directional antennas may be useful for very low power terminals or for an occasional long range shot to a predetermined location. It is expected that alignment will involve greater installation complexity and coordination and, in addition to taking somewhat longer to install, may rule out simple forms of deployment. Assuming no problems are encountered with the equipment as a result of the transportation, it should be possible for a small team of persons to completely install the field elements of a Radionet (or remove them) in little more time than it takes to reach the appropriate deployment locations.

Unattended operation and reliability considerations

Once in place, a packet radio network should not require the presence of any personnel for its normal operation. Furthermore, certain debugging and restart or shutdown operations should be possible to perform remotely. Power consumption should be kept to a minimum and reduced to almost zero during periods when no packets are being processed. This technique will lengthen the interval between maintenance visits which is expected to be principally determined by the powering requirements.

A hand-held terminal might be expected to provide several hours of service without recharging. A repeater unit should be able to provide service for many days, if not several weeks or months between replenishments of the power supply. This time period will depend on the amount of traffic handled.

The probability that some part of the network is unable to communicate with the rest of the net should be less than 0.5 percent and the mean time between failures should be at least 1000 hrs for any component in the field under normal conditions.

Error free performance

The system should provide essentially error-free performance for computer communications. A target objective of no more than one undetected packet error per 10exp(10) packets assuming 1000 bit packets, a 100 kilobit/second data rate and 100 percent occupancy. This is equivalent to a mean time between undetected packet errors in excess of 30 years.

Experiments in urban areas have shown[10] that noise impulses occur every few milliseconds in both the UHF and L bands principally due to automobile ignition noise. A packet has a very high probability of encountering one or two impulses and therefore some form of error correction is required. In general the error correction choice should depend on the characteristics of the environment. A simple form of error correction should be provided and the system design should facilitate the addition of additional error correction capability, if necessitated by the environment.

Internetting

Internetwork communication is of particular importance in the computer communications field due to the expected proliferation of multiple nets and the high cost of user interfacing. Thus, it may be necessary for a user or a Host on the Radionet to access a particular user or resource on some other packet switched network. This should be possible using a protocol designed for internetting so that differences in format, packet size, addressing and other conventions can be properly handled.[9]

Resource allocation

When traffic levels are sufficiently low, there is little need for regulating or otherwise controlling the use of the packet radio channel. As the potential demand for service rises, however, a point is reached where some form of system control of the channel is necessary for resource allocation. The packet radio system must be able to guarantee service to authorized individual subscribers so that their requirements are provided by the system even when certain users require or are privileged to obtain a larger allocation of resources than others.

Directories and virtual subnets

A user on the Radionet should be able to communicate with other users or systems by name rather than strictly by numerical ID. Thus, a user named Smith could be reached using only Smith's alphanumeric name. The system performs the directory lookup and all necessary conversions from Smith to the appropriate internal addressing on behalf of the user who need never be aware of the address mapping. If more than one Smith were known to the system, the user would have to supply additional information to distinguish among them. Smith may also choose to refuse communications.

In addition, selected users on the net may wish to form logical subnets of their own within the Radionet for handling specific functions or for otherwise organizing themselves. The net should provide a mechanism to allow the formation of these virtual subnets and allow authorized individual users to join or leave these subnets as needed. Initially, these subnets could be provided as a special service to the net by one or more Hosts on the network.

SYSTEM STRUCTURE

The communication system consists of a distributed array of packet radio repeaters (see Figure 5) each of which

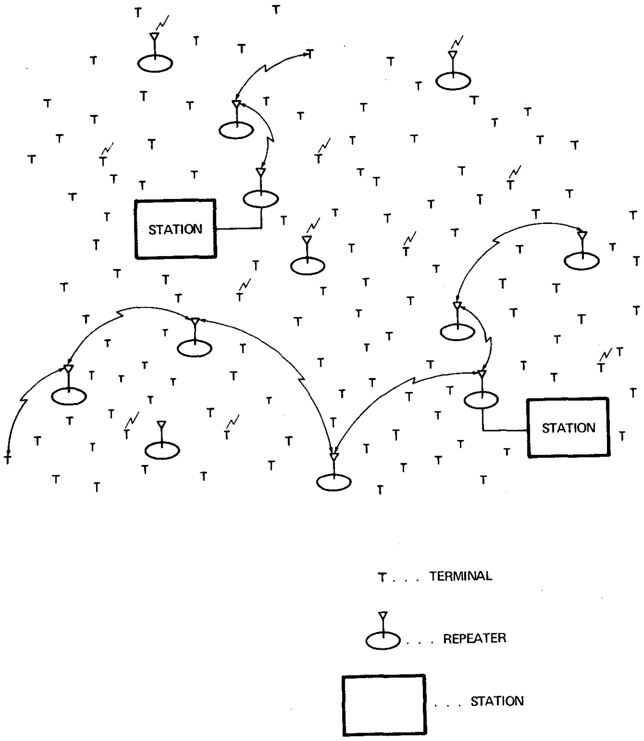


Figure 5—User communication in a packet radio network

is able to receive and then transmit a sequence of packets, thereby serving as a relay. The range of each repeater is determined by its geographic setting and its effective radiated power. A maximum spacing of 20-25 miles between repeaters is practical in the light of the target requirements in the previous section. A denser packing of repeaters is desirable for communication with hand-held terminals and for reliability. In particular, certain repeaters could be placed in the field and remain inactive until needed for reliability.

The initial system concept utilizes a single radio channel shared by all the repeaters which operate as transceivers. Two repeaters may communicate with each other by "leapfrogging" over any other repeaters which may happen to lie in their path. A discussion of these choices is contained in the companion paper by Frank.[2] Simulation results show that a higher data rate (e.g., 400 kilobits/second) should be used for repeater to repeater communication than for terminal to repeater communication (100 kilobits/sec).

The 1710-1850 MHz fixed and mobile band has been selected for the initial experimentation. The system will be operated in a 20 MHz portion of the band. In addition to being wide enough to support the initial requirement of 20 MHz, it is capable of supporting possible requirements for multiple channel operation using a 100 MHz bandwidth. Alternate bands in the range between 100 MHz and 2 GHz were considered for possible selection, but were rejected for the initial tests due to their heavy usage. Operational use of the Radionet at some lower frequency can be easily achieved by using a different amount of band shifting.

Although the characteristics of the lower frequency bands are somewhat better suited to radio propagation in adverse environments, the higher bands will afford a more conservative system test. The use of the lower frequencies is particularly important when all users are expected to receive directly the initially launched wave which may encounter built-up areas, mountains, etc. For a repeater system, it makes less difference if the waves pass around mountains since the end user would not normally be expected to receive the initial wave, but only a repeater version of it. Although it is slightly more efficient to generate power at the lower frequencies, this difference in the bands under consideration is no greater than a factor of two and is compensated for by the scaling down in size of components, particularly the antenna.

As there is a potential for confusion between the function of a repeater and the function of a terminal, we digress a moment to discuss terminology. A device which is capable of transmitting a radio packet or receiving a radio packet is called a packet radio. A repeater, in our context, is merely a particular kind of packet radio which is equipped to retransmit by radio some or all packets which it receives by radio.

A terminal on the Radionet also requires the capability to transmit radio packets and to receive radio packets. Its role, though, is one in which it accepts no packets destined for other users and consequently it only transmits packets which are originated by its user. That is, it does not act as a repeater and it is not programmed to act as a repeater. A repeater is simply a packet radio which happens to be programmed to function in a repeater mode. If there is need for clarification, we shall refer to the packet radio at the terminal, or to the packet radio repeater (or simply repeater).

Practically speaking, there will be some difference in implementation between a repeater and a packet radio at the terminal, although they will both function in similar (if not identical) ways. For one, the repeater will be more heavily powered to last longer and may be a more rugged unit (particularly if it must be housed outdoors in all weather). A good description of the technology for packet radio is given in Reference 1.

A packet radio which fits in one's pocket will obviously have different attributes. In addition, the user will require some means of inputting data to the packet radio and reading out received data. For example, this may take the form of voice or keyboard input, and display, print or voice output.

The logical functions of packet processing in the Radionet are handled by microprocessors. A user terminal may actually have two separate microprocessors, one for keyboard and display control and one for the radio processing, or it may have only one to provide both functions. A discussion of digital terminals for packet radio networks is given in a companion paper.[3]

The repeaters are designed to be relatively simple, and most of the system control functions are deliberately separated out for handling by one or more control stations. The control station consists of a minicomputer which is a repository of centralized information and control for the packet Radionet. It is connected to the rest of the system by its own packet radio. The packet radio at the station is logically equivalent to a packet radio at a terminal. However, the radio at the station may also serve as a repeater in the multi-station case.

For reliability, a packet Radionet will contain two or more stations. Two ways to allocate responsibilities between the stations are:

(1) To have one station assume the active role of controller while the other stations are kept in readiness and advised of the network status in order that one of them may automatically become active should the currently active control station fail or need to be removed from service.

(2) To dynamically apportion or partition the control task among the stations such that each station only serves its share of the users and/or network resources. Backup of each station by the others is also provided in this case and automatic switchover occurs if a station fails.

A goal in either case above is to quickly achieve a switchover with as little loss of data as possible (hopefully none). Both of these techniques are currently under study. A description of the station and its functions is given in Reference 3.

In the initial tests, all data will flow from the user to the station and then from the station to the final destination! However, the protocols are such that it would be possible to support direct communications between source and destination for some or all of the network communication (with the station not in the path). However, changes in the repeater functions may also be necessary if the station is not directly involved, which might complicate the repeater design. These alternatives will be a subject for study during the test program to determine whether or not (1) it is desirable or necessary for the station to be in the path to guarantee allocation of resources or to provide critical services for certain users, (2) to determine the limits on system performance induced by the station in the path, and (3) to determine how the results of (1) and (2) above are affected if multiple stations are simultaneously in use within the same net. The station also serves as the gateway to other networks, and unless the destination user or Host resource is also on the packet Radionet, all traffic must pass through a station anyway.

When the station is in the direct path, it provides flow control. Each packet is hop by hop acknowledged between repeaters for error control as it proceeds to the station and from the station, but is acknowledged from the station to the packet radio at the terminal and from the destination to the station for flow control. A Host is allowed to have multiple packets in transit within the net at any time. End to end acknowledgments are required every N packets according to the flow control doctrine, and do not depend on the proper sequential receipt of packets by the station. The Radionet will allow occasional packets to be delivered out of order unless specifically required to provide sequencing for a given terminal or Host. It is assumed that the Host/Host protocol is prepared to handle sequencing.

A user's packet radio makes initial contact with the station by sending a search packet. All repeaters which hear the search packet forward it to the station which, in turn, selects one of the repeaters and transmits the address of the selected repeater to the users packet radio. As a user moves about within the boundaries of the Radionet, the station detects when a handoff from one repeater to another is desirable and performs the handoff by advising the user's packet radio to address a new repeater.

The repeaters do not determine routes. All the routing computations are performed by the station. The complete routing information for each packet is inserted in the packet by the source and is carried along with the packet as it moves through the Radionet.

AUTHENTICATION AND PRIVACY

In principle, every packet radio is capable of being modified to receive packets not intended for it, and packets must be encrypted to provide privacy or security. The situation in the Radionet is considerably different from that which occurs in a wireline network where access to internal network packets is restricted. Similarly, every receiver in a Radionet is also a transmitter and capable, in principle, of being modified to masquerade as another packet radio. Some mechanism must therefore be introduced into the net to verify the authenticity of both the sender and the receiver in each stage of the communication process, and to avoid disruption of normal communications.

The same basic considerations apply whether we focus on a repeater or on a packet radio at the terminal. Consequently, in the following discussion, we shall only consider the case of authenticating repeaters within the net. Authentication and privacy mechanisms will not be incorporated into the initial packet radio system, but are planned for incorporation in a later phase.

Authentication

Let us assume that all data is suitably encrypted for privacy and that we are interested in verifying at repeater A that traffic leaving repeater A and addressed to repeater B is actually being received by intended repeater B. In addition, we are interested in verifying that traffic being received from a so-called repeater B is actually coming from an authentic repeater B.

One possible approach to this problem is outlined below. Each repeater is assumed to include an algorithm whose operation is unknown to the users and which may be changed dynamically. Identification of non-authentic repeaters is primarily dependent upon the detection of a violation of protocol or by observing a repeater using the wrong algorithm. The algorithm is assumed to be packaged in a repeater in such as way that it cannot be read out if a repeater is captured. We recognize that this assumption, as well as the assumption that the algorithm can be kept private enough, may be subject to question. However, the packaging of the algorithm in an appropriate way is believed to be achievable and an approach of this form is believed to be necessary if mobile terminals and unattended operation of repeaters is to be achieved in the Radionet.

Several methods of using the algorithm are currently under study. One possible method involves a three way handshake to communicate data. Let us assume that repeater A selects a number at random and communicates it to repeater B with an implicit request to forward a packet. Repeater B, acting suspiciously, uses the received number to generate a new number with the aid of the algorithm. Repeater A also generates the new number which it uses to select a spread spectrum code pattern. Repeater B acknowledges the request using the spread spectrum code. Repeater B then prepares to receive the data within several milliseconds after receiving the original request. The request thus serves as the preamble for signalling the arrival of a packet. It could also provide exact timing for the data, if desired.

The data is sent by repeater A using another portion of the spread spectrum code. The acknowledgment from B to A serves to validate the receiver to A and the receipt of the data by B from A in the correct code serves to validate A to repeater B. Eventually the packet will be correctly received by B and, to avoid endless repetitions from A, B acknowledges the correct receipt back to A using yet another portion of the spread spectrum code. One effect of this handshake procedure, however, is to reduce the capacity of the system and to increase the average delay in order to provide authentication and to protect the transmission from unwanted interference. Several techniques are available for insuring that some postulated level of interference can be overcome, through the use of spread spectrum for anti-jam protection and by random selection of preambles.

Privacy

Data is encrypted at the point of origination and decrypted only at the final destination. Although we do not discuss it further in this paper, it is assumed that a key distribution scheme is used to achieve the requisite protection. In Figure 6 we illustrate a terminal, a station and a Host on the Radionet. The terminal is shown with two microprocessor units, one on either side of an encryption unit. One microprocessor is an integral part of the packet radio. The other microprocessor is used to support the keyboard display portion of the terminal.

Each packet is assumed to be encrypted independently of the others. Only one encryption unit is shown at the Host, but for multiple access, more than one might be desired. In this situation, the station only serves to provide functions such as flow control, resource allocation, and address mapping. It does not and cannot interpret the data which passes by in any meaningful way.

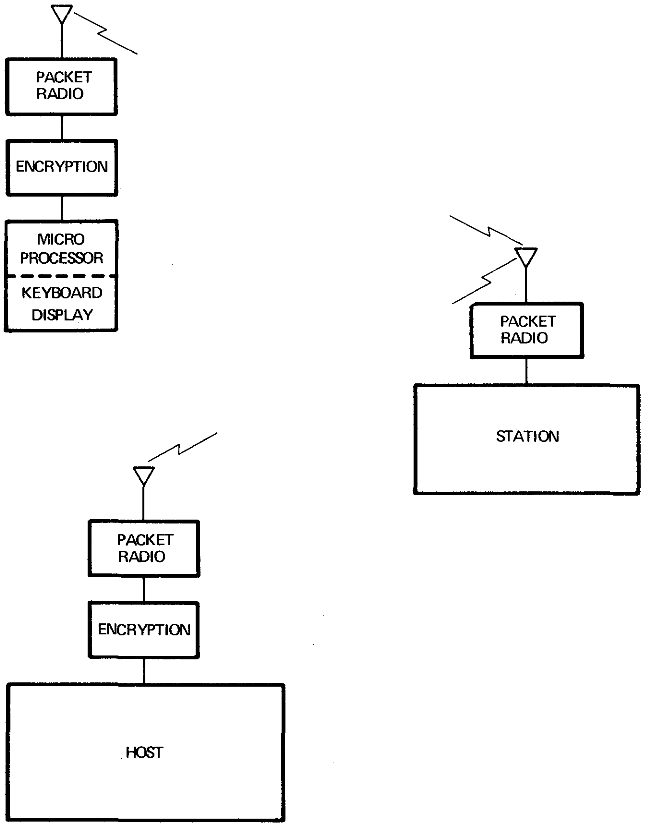


Figure 6—Microprocessors and encryption

Data from the terminal is packetized by its microprocessor prior to encryption where it may also be echoed according to a Telnet protocol.[20] We assume the address of the desired destination Host is resident at the station, it having been notified prior to setup of the connection either by the terminal (or the key distribution system). The encrypted packet is stored in the packet radio microprocessor until acknowledged by the next repeater. The station may sequence packets from the terminal, but is not assumed to be able to ask for retransmissions. Such requests are assumed to come as part of the end to end protocol with the Host or other user. Communication from the Host to the terminal is typically sequenced by the station to simplify the reassembly task at the terminal. There is no such requirement in general for sequencing of packets headed to the Host. In certain cases, such as with speech, there may be no advantage to reordering packets headed to the terminal either.

In internetwork connections between the Radionet and another net, the station serves as a gateway. However, the encrypted packets are forwarded as before to the destination Host without any meaningful interpretation at the station, and none is possible.

SWITCHING AND SORTING APPLICATIONS

The use of packet radio in switch design was briefly discussed at the beginning of the first section. Its use in the implementation of an mxn crossbar switch with $m+n$ radio units is also clear. In fact, if the output lines from the switch coincide with the input lines, then only m radio units are needed. Another example of the use of packet radio in bucket sorting is discussed below.

A bucket sort is one way of processing data items for rapid storage and retrieval from a small table. Let us assume that M data items are to be stored in K buckets. A unique name must be supplied to retrieve a data item. Each data item (and the unique name) is assigned to one of the buckets based on a simple test (e.g., its low order n bits).

In a typical implementation of a bucket sort, all the bucket items are stored in a single processor. Entries within a bucket are chained together serially. To store an item, its bucket is first identified and the item is appended to the end of the chain either by searching for the end or with the aid of a pointer to the end. To retrieve the item, a search of the chain is required. On the average, this takes M/K tries.

Now consider the implementation of a bucket sort on a set of microprocessors all in communication with each other via packet radio. The buckets are now conceptual entities each of which is distributed among the microprocessors. For simplicity, each processor is assumed to store only one entry from any single bucket, but may also store entries from other buckets. If bucket i contains k entries, they would be stored in k distinct microprocessors. Responsibility for storing the next item in a bucket is passed around round robin among the processors so that, at any time, only one microprocessor is designated to store the next item for a given bucket. Once that item is stored, some other microprocessor is designated to store the succeeding item. The storage contents of a set of microprocessors is shown schematically in Figure 7. The items in bucket 2 are shown in boldface on the figure.

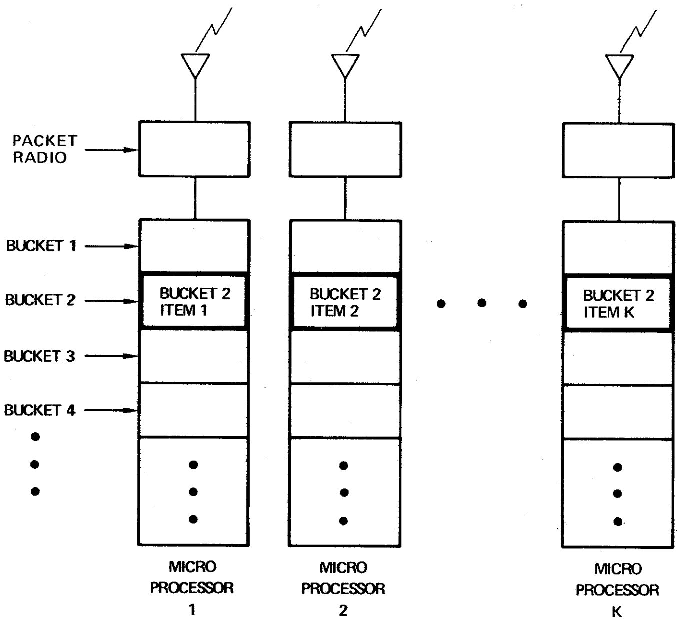


Figure 7—Bucket sorting by packet radio

In order to retrieve an item in bucket j, a microprocessor broadcasts the unique name. Each microprocessor hears the request, determines the bucket and performs a match with the contents of the j-th entry in its table. If the contents match the unique name, it broadcasts back the requested data item and the unique name. A data item may therefore be retrieved in exactly one try. If the number of microprocessors does not exceed the number of entries per bucket, the number of tries will be greater than one, but still much less than M/K.

Many other applications of packet radio can be envisioned including remote monitoring and detection networks, Satellite networks, factory automation, linguistics, and command and control. However, no attempt is made to discuss these or other applications in this paper.

CONCLUSIONS

The packet radio technology utilizes a distributed set of microprocessors to provide computer control of a multiple access radio communication system. It is capable of supporting switched wideband communications over very short distances (inches) and over wide geographic areas (hundreds of miles). Within limits, the system allows coexistence with other, possibly different, systems which may reside in the same frequency band.

An overview of the initial packet radio system was presented in this paper. Several of its design objectives are expected to be upgraded in a latter phase of the system development. These include higher data rates, smaller repeater size, expanded station functions, and improved coding and reception techniques for fading, interference, and multi-access channels. Authentication and privacy mechanisms will be incorporated at that time.

Among the wide variety of uses to which packet radio may be put are the following:

1. Personal radio terminals—Each user has his or her personal radio terminal which he or she can use to communicate over the radio system with other subscribers and resources. Although keyboard entry and display devices are envisioned for the initial experimental terminals, low rate, high quality speech input/output devices will soon be available as viable means of computer interaction, and could augment the initial terminals.
2. Cable TV—The basic concept of packet radio may be applied to two way cable TV systems for use within buildings and in urban areas. In principle, the radio signal (prior to RF conversion) could be sent over one or more Cable TV channels.
3. Computer Architecture—Low power, low cost packet radio devices in small enough size would allow the assembly of wireless combinations of computer resources such as memory, processors and I/O

devices. Such structures could provide for effective organization of large scale computational procedures. We believe that packet broadcasting may become a natural way for system architects to interconnect large numbers of microprocessors in the future.

4. Rapid Deployment—It is often desirable to introduce communications to a temporary work site in a very short time period. Packet radio is ideally suited to this requirement. In fact, the technology may herald an era of discardable electronics if the technology can be made low enough in cost and size.
5. Frequency Management—Spectrum management techniques are sorely needed to assure that the present and expected requirements for use of the frequency bands can be effectively satisfied. Computer control of the spectrum along with coexistence techniques will allow graceful transitions from inefficiently used single user systems to more efficiently used multiple access systems.

REFERENCES

1. Fralick, S. and J. Garrett, "A Technology for Packet Radio," *AFIPS Conference Proceedings,* Volume 44, 1975, AFIPS Press, Montvale, N.J.
2. Frank, H., I. Gitman and R. VanSlyke, "Packet Radio Network Design—System Considerations," *AFIPS Conference Proceedings,* Volume 44, 1975, AFIPS Press, Montvale, N.J.
3. Fralick, S., D. Brandin, F. Kuo and C. Harrison, "Digital Terminals for Packet Broadcasting," *AFIPS Conference Proceedings,* Volume 44, 1975, AFIPS Press, Montvale, N.J.
4. Burchfiel, J., R. Tomlinson and M. Beeler, "Functions and Structure of a Packet Radio Station," *AFIPS Conference Proceedings,* Volume 44, 1975, AFIPS Press, Montvale, N.J.
5. Kleinrock, L. and F. Tobagi, "Random Access Techniques for Data Transmission over Packet Switched Radio Channels," *AFIPS Conference Proceedings,* Volume 44, 1975, AFIPS Press, Montvale, N.J.
6. Binder, R., et al., "Aloha Packet Broadcasting—A Retrospect," *AFIPS Conference Proceedings,* Volume 44, 1975, AFIPS Press, Montvale, N.J.
7. Abramson, N., "Another Alternative for Computer Communications," *AFIPS Conference Proceedings,* FJCC70, pp. 695-702.
8. Roberts, L., "Extension of Packet Switching to a Hand Held Personal Terminal," *AFIPS Conference Proceedings,* SJCC72, pp. 295-298.
9. Cerf, V. and R. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Transactions on Communications,* May 1974, pp. 637-648.
10. Nielson, D., *Microwave Propagation and Noise Measurements for Digital Radio Application,* SRI Internal Report, February 1975.
11. Roberts, L., "Data by the Packet," *IEEE Spectrum,* February 1974.
12. Kino, G. S. and H. Matthews, "Signal Processing in Acoustic Surface-wave Devices," *IEEE Spectrum,* August 1971, pp. 22-35.
13. McConoughey, S. R., "New Concepts in Spectrum Usage," *IEEE Transactions on Communications,* Vol. COM-21, Nov. 1973, pp. 1172-1176.
14. Kleinrock, L., "Analytic and Simulation Techniques in Computer Network Design," *AFIPS Conference Proceedings,* SJCC70, pp. 569-579.
15. Kleinrock, L. and F. Tobagi, "Carrier Sense Multiple Access for Packet Radio Channels," *Proceedings of the International Conf. on Communications,* Minneapolis, Minn., June 1974.
16. Cover, T., "Broadcast Channels," *IEEE Transactions on Information Theory,* Jan. 1973, pp. 2-14.
17. Slepian, D. and J. Wolf, "A Coding Theorem for Multiple Access Channels with Correlated Sources," *Bell System Technical Journal,* September 1973.
18. Liao, H., *Multiple Access Channels,* Ph.D. dissertation, Department of Electrical Engineering, University of Hawaii, Honolulu, Hawaii, 1972.
19. "Special Issue on Microwave Acoustic Signal Processing," *IEEE Trans. on Microwave Theory and Techniques,* Vol MTT-21, April 1973.
20. *Remotely Controlled Telnet Echoing,* RFC 581, Network Information Center, Stanford Research Institute, Menlo Park, Calif.
21. Heart, F., et al., "A New Minicomputer/Multiprocessor for the ARPA Network," *AFIPS Conference Proceedings,* NCC73, pp. 529-537.
22. Network Analysis Corporation, Glen Cove, New York, *Semiannual Technical Report,* 1975.
23. Roberts, L. G. and B. D. Wessler, "Computer Network Development to achieve Resource Sharing," *AFIPS Conference Proceedings,* SJCC70, pp. 543-549.
24. Kahn, R. E., "Resource Sharing Computer Communication Networks," *IEEE Proceedings,* Nov. 1972, pp. 1397-1407.
25. Abramson, N., "Packet Switching via Satellite," *AFIPS Conference Proceedings,* NCC73, pp. 696-702.
26. Kleinrock, L. and S. Lam, "Packet Switching in a Slotted Satellite Channel," *AFIPS Conference Proceedings,* NCC73, PP703-710.
27. Heart, F. and R. Kahn et al., "The Interface Message Processor for the ARPA Computer Network," *AFIPS Conference Proceedings,* SJCC70, pp. 551-567.
28. Farber, D. and K. Larson, "The Structure of a Distributed Computer System—Software," *Proceedings of the Symposium on Computer-Communications and Teletraffic,* Polytechnic Press, MRI Symposium Proceedings, Vol. XXII, 1972, kpp. 539-545.
29. Frank, H., W. Chou and I Frisch, "Topological Considerations in the Design of the ARPA Computer Network," *AFIPS Conference Proceedings,* SJCC70, pp. 581-587.
30. Frank, H., R. Kahn, and L. Kleinrock, "Computer Communication Network Design—Experience with Theory and Practice," *AFIPS Conference Proceedings,* SJCC72, pp. 255-270.
31. Jackson, P. and C. Stubbs, "A Study of Multi-Access Computer Communications," *AFIPS Conference Proceedings,* SJCC69, pp. 491-504.
32. Roberts, L. G., "Dynamic Allocation of Satellite Capacity through Packet Reservation," *AFIPS Conference Proceedings,* NCC73, pp. 711-716.
33. Lam, S. and L. Kleinrock, "Packet Switching in a Multi-Access Broadcast Channel; Dynamic Control Procedures," Submitted to *IEEE Transactions on Communications.*
34. Kleinrock, L., W. Naylor and H. Opderbeck, "A Study of Line Overhead in the ARPANET," to be published in *Communications of the ACM,* 1975.
35. Kleinrock, L., "Resource Allocation in Computer Systems and Computer Communication Networks," *IFIP Congress Proceedings,* IFIP74, pp. 11-18, North Holland Publishing Co., 1974.
36. Abramson, N. and F. Kuo (co-editors), *Computer-Communication Networks,* Prentice-Hall Publishing Company, 1973.
37. "Special Issue on Mobile Radio Communications," *IEEE Transactions on Communications,* Vol. Com-21, November, 1973.

Random access techniques for data transmission over packet-switched radio channels*

by LEONARD KLEINROCK and FOUAD TOBAGI
University of California
Los Angeles, California

INTRODUCTION

Terminal access to computer systems has long been and continues to be a problem of major significance. We foresee an increasing demand for access to data processing and storage facilities from interactive terminals, point-of-sales terminals, real-time monitoring terminals, hand-held personal terminals, etc. What is it that distinguishes this problem from other data communication problems? It is simply that these terminals tend to generate demands at a very low duty cycle and are basically *bursty* sources of data; in addition, these terminals are often geographically distributed. In the computer-to-computer data transmission case, one often sees high utilization of the communication channels; this is just not the case with terminal traffic. Consequently, the *cost* of providing a dedicated channel to each terminal is often prohibitive. Instead, one seeks ways to merge the traffic from many terminal sources in a way which allows them to share the capacity of one or a few channels, thereby reducing the total cost. This cost savings comes about for two reasons: first, because of the economies of scale present in the communications tariff structure; and secondly, because of the averaging effect of large populations which permit one to provide a channel whose capacity is approximately equal to the sum of the *average* demands of the population, rather than equal to the sum of the *peak* demands (i.e., the law of large numbers). This merging of traffic and sharing of capacity has been accomplished in various ways such as: polling techniques, contention systems, multiplexing, concentrating, etc. Many of these are only weak solutions to the problem of gathering low data rate traffic from sources which are geographically dispersed.

In this set of papers,[1-6] we suggest another solution to the terminal access problem, namely that of packet switching over radio channels. In such a system, data terminals package their data into constant length segments known as packets to which is added additional control information such as source and destination address, error control bits, etc. All terminals are assumed to share a common (wide-band) radio channel and to be within range and in line-of-sight of a receiver station. When any terminal generates a packet, that terminal follows some transmission protocol which determines when transmission may take place at which time the packet is transmitted using the full channel bandwidth. Depending upon the protocol, more than one terminal might (unfortunately) transmit in overlapping time intervals, in which case these packets may destructively interfere with each other. Whenever the station receives a packet correctly (as determined by the error control sum check), then an acknowledgment is broadcast to the terminal population, identifying which packet was correctly received. If a terminal receives no acknowledgment after some appropriate timeout interval, then it knows that its packet was "destroyed" and must take some action to cause a retransmission attempt. The key point is that all terminals are simultaneously sharing a single channel; this offers a solution which handles the geographical dispersion of terminals and which at the same time takes advantage of the available cost savings mentioned earlier. Moreover, this solution is highly effective when terminals are mobile (police cars, fire trucks, taxis, ambulances, army vehicles and personnel, etc.) and/or when the environment is itself hostile (natural dangers or man-made dangers).

The use of radio packet switching is relatively new* and has been reported upon in the recent literature. The ALOHA system[7] at the University of Hawaii is not unlike the system we have in mind, and the description of experience with this system as it impacts the current study is described in these proceedings.[2] In 1973, a series of papers describing the use of packet switching in satellite radio channels was published in these proceedings;[8-10] the satellite problem is very similar to the terminal radio problem, with the key distinction being the enormous difference in the propagation delay (roughly ¼ second for a stationary satellite as opposed to small fractions of a millisecond for line-of-sight ground radio).

The Advanced Research Projects Agency of the Department of Defense, recently undertook a new effort whose goal is to develop new techniques for packet radio communication

* This work was supported by the Advanced Research Projects Agency of the Department of Defense (DAHC-15-73-0368).

* On the other hand, digital (pulse) systems using radio propagation are not new—e.g., telegraphy, radar, etc. Here we restrict our comments to addressed packets. The most well known example of a packet switched wire network is the ARPANET.[11]

among geographically distributed, fixed or mobile, user terminals and to provide improved frequency management strategies to meet the critical shortage of r.f. spectrum. The research presented in this paper is an integral part of the total design effort of this system which encompasses many other research topics. A number of these are considered in this set of papers. In this paper, we are concerned with *one* aspect of design and analysis, namely the consideration of various random access protocols, their behavior, and the difficult problem of controlling a channel which must carry its own control information. Specifically, we do *not* investigate the networking issues when radio relays (repeaters) are required to extend the range of the terminals; such issues (layout, routing, etc.) are dealt with in References 1 and 4. We consider an environment in which all terminals are within radio range and line-of-sight of a common receiver station. One of the first protocols studied in conjunction with ground radio and satellite packet switching was "pure ALOHA" as mentioned above. In this mode, users are permitted to transmit any time they desire. If they receive an acknowledgment within some predetermined time-out period, then they know their transmission was successful. Otherwise they assume a multi-access collision occurred and they must retransmit. To avoid the same collision again (and forever!) any one of many schemes may be used for introducing a random retransmission delay, thereby spreading the conflicting packets over time. It is known that the maximum fraction of successful packet transmissions on the average is simply $\frac{1}{2}e(\approx 18$ percent) for random ALOHA.[7] This is abominably small compared to the maximum of 100 percent successful if transmission were perfectly scheduled to avoid all collisions. A second method for using the radio channel is to modify the completely unsychronized use of the ALOHA channel by "slotting" time into segments whose duration is exactly equal to the transmission time of a single packet (assuming constant length packets). If we require each user to start his packets only at the beginning of a slot, then when two packets conflict, they will overlap completely rather than partially, providing an increase in channel efficiency. This method is referred to as "slotted ALOHA."[9] The optimum performance of this system is twice that of random ALOHA, namely $1/e(\approx 37$ percent); this is still poor. Not only is the *capacity* of the ALOHA channels wanting, but so too is the average delay $\mathfrak{D}$ until successful transmission; we give the throughput-delay characteristic later in Figure 7.

Let us compare slotted ALOHA to Frequency Division Multiple Access (FDMA) which is a common method for partitioning a channel into a given number of separate subchannels which are assigned on a point-to-point basis between user pairs; synchronous Time Division Multiple Access (TDMA) is equivalent to FDMA so far as we are concerned here (we neglect guard bands). The fixed channel assignment in FDMA is effective in preventing collisions but succeeds in this at the expense of possibly poor utilization of each channel since the smoothing effect of a large population is absent. To analyze FDMA, we adopt the following assumptions: (a) an assumed finite (but large) population of M users; (b) each user generates a new fixed length packet (of b_m bits) according to a Poisson process at a rate Λ per second; (c) the total channel has a bandwidth of W hertz modulated at 1 bit/hertz-sec (giving a channel capacity of W bits/sec). Thus, with M users in this FDMA mode, each is assigned a channel of W/M bits/sec. Each such channel behaves as an $M/D/1$ queueing system giving an average time in system $\mathfrak{D}$ (waiting plus transmission) as follows:[12]

$$\mathfrak{D}=\frac{\rho\left(1-\frac{\rho}{2}\right)}{1-\rho} \tag{1}$$

where $\rho = Mb_m/W$.

We are assuming that queueing is permitted at each terminal. However, the analysis for slotted ALOHA assumes an infinite population of users with an aggregate input rate of $M\Lambda$ packets per second and this produces an upper bound on delay. (We note that a finite population model with M users at rate Λ and with queueing permitted will produce fewer collisions than the infinite population would since each terminal will avoid conflicts among its own packets).

Equation (1) for FDMA is compared with the results for delay in slotted ALOHA with an infinite population (see Reference 8 and Figure 7 below) as follows. We consider the (M, Λ) plane in Figure 1, in which we represent constant $\mathfrak{D}$ contours. Comparing the delay performance of the two systems, we note that when we are in presence of bursty users (small Λ), slotted ALOHA can support many more users than FDMA, for the same packet delay. For example, at $\mathfrak{D}=0.1$ sec, slotted ALOHA can support a number of users which is over 3 orders of magnitude greater than the number that FDMA can support when $\Lambda=10^{-3}$ packet/sec; as Λ increases (i.e., as the burstiness decreases), this difference reduces until at $\Lambda\approx 5$ the two systems can support roughly an equal number of users. Beyond this point, FDMA is superior. This crossover point clearly depends upon the value of $\mathfrak{D}$ examined. In fact, slotted ALOHA can support total traffic only in the range $M\Lambda b_m/W<1/e\cong.37$ and beyond

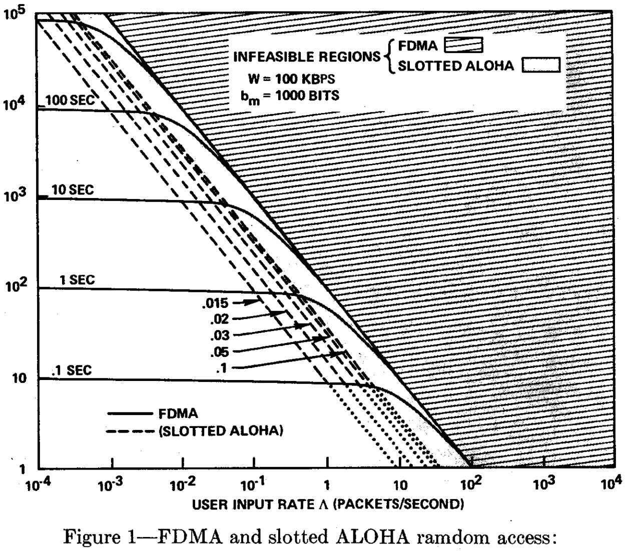


Figure 1—FDMA and slotted ALOHA ramdom access: Performance with 100 KBPS bandwidth

that, FDMA will always be superior until it too saturates at $M\Lambda b_m/W=1$; this tradeoff is clearly evident in the curves of Reference 10.

The above result can be alternatively presented in the following manner. Let M be some large number, say 1000. Figure 2 shows constant $\mathcal{D}$ contours in the (W, Λ) Plane. Again we note that if we are in presence of bursty users, in order to achieve the same small delay, FDMA requires a bandwidth larger than slotted ALOHA by as much as three orders of magnitude. This factor is exactly equal to M as $\Lambda \rightarrow 0$ since in this region queueing effects are insignificant; in this limit the delay $\mathcal{D}$ is simply the packet transmission time (observe the flatness of the curves in Figures 1 and 2), which for FDMA is $\mathcal{D}=Mb_m/W$ and for slotted ALOHA is $\mathcal{D}=b_m/W$. It is also obvious here, for the same total bandwidth W, that FDMA will give M times the delay as compared to slotted ALOHA. This gain diminishes as Λ increases, until finally as $M\Lambda b_m/W \rightarrow 1/e$ the situation reverses as mentioned above.

Finally, let us fix Λ and consider the delay contours in the (W, M) plane. Figure 3 corresponds to $\Lambda=10^{-1}$ packets per second. Such input rates correspond again to bursty users. We note again that in order to support a large number of users, FDMA requires a larger bandwidth for the same delay performance.

It is all too evident from the above comparison that random access is by far superior to FDMA or TDMA when the environment consists of large populations of bursty users. However, we note that slotted ALOHA itself does not use the channel as efficiently as we might hope and this prompts one to inquire as to other, superior, protocols; such an inquiry is the subject of this paper. Following we consider two random access modes which we refer to as "Carrier Sense Multiple Access" (CSMA) and "Split-channel Reservation Multiple Access" (SRMA).

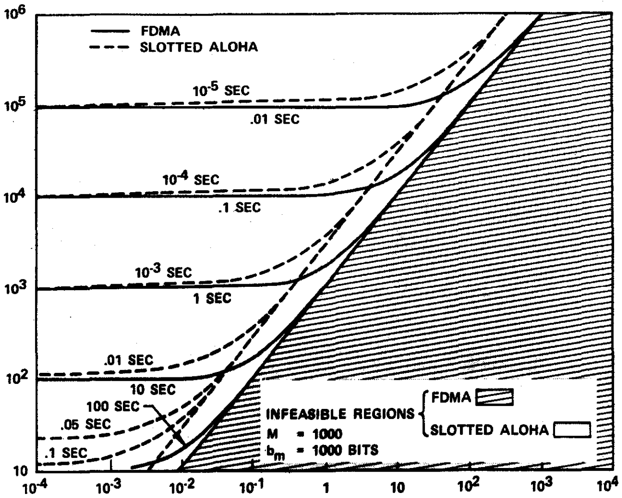


Figure 2—FDMA and slotted ALOHA ramdom access: Bandwidth requirements for 1000 terminals

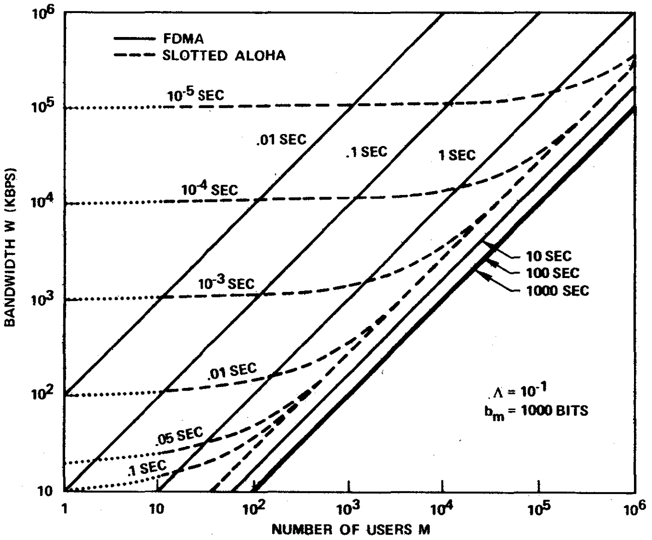


Figure 3—FDMA and slotted ALOHA ramdom access: Performance for $\Lambda=10^{-2}$ packets per second

CARRIER SENSE MULTIPLE ACCESS MODES

The radio channel considered in this paper is characterized as a wideband channel with a propagation delay between any source-destination pair which is very small compared to the packet transmission time.[*] This suggests a new approach for using the channel; namely, the Carrier-Sense Multiple Access (CSMA) mode. In this scheme one attempts to avoid collisions by listening to (i.e., "sensing"[†]) the carrier due to another user's transmission. Based on this information about the state of the channel, one may think of various actions the terminal may take. Three protocols will be considered which we call "persistent" CSMA protocols: the 1-Persistent, the Non-Persistent, and the p-Persistent CSMA. Below, we present the protocols, and display the throughput-delay performance for each.

In this paper we omit the proofs for conciseness and clarity of presentation; the details of these proofs are to be found in a series of forthcoming papers.[13–15]

CSMA transmission protocols and system assumptions

The various protocols considered below differ from one another by the action (pertaining to packet transmission)

[*] Consider, for example, 1000 bit packets transmitted over a channel operating at a speed of 100 Kilobits per second. The transmission time of a packet is then 10 mseconds. If the maximum distance between the source and the destination is 10 miles, then the (speed of light) packet propagation delay is of the order of 54 μseconds. Thus the propagation delay is a very small fraction (a = 0.005) of the transmission time of a packet. On the contrary, when one considers satellite channels [8] the propagation delay is a relatively large multiple of the packet transmission ($a \gg 1$).

[†] Sensing carrier prior to transmission is a well-known concept in use for (voice) aircraft communication. In the context of packet radio channels it was originally suggested by D. Wax of the University of Hawaii in an internal memorandum dated March 4, 1971.

that a terminal takes after sensing the channel. However, in all cases, when a terminal determines (by the absence of a positive acknowledgment) that its transmission was unsuccessful, then it reschedules the transmission of the packet according to a randomly distributed retransmission delay. At this new point in time, the transmitter senses the channel and repeats the algorithm dictated by the protocol. At any instant a terminal is called a *ready terminal* if it has a packet ready for transmission at this instant (either a new packet just generated or a previously conflicted packet rescheduled for transmission at this instant).

A terminal may, at any one time, either be transmitting or receiving (but not both simultaneously). However, the delay incurred to switch from one mode to the other is negligible. All packets are of constant length and are transmitted over an assumed noiseless channel (i.e., the errors in packet reception caused by random noise are not considered to be a serious problem and are neglected in comparison with errors caused by overlap interference). The system assumes non-capture (i.e., the overlap of any fraction of two packets results in destructive interference and both packets must be retransmitted). We further simplify the problem by assuming the propagation delay τ (small compared to the packet transmission time) to be identical* for all source-destination pairs.

1-Persistent CSMA

The 1-Persistent CSMA protocol is devised in order to (presumably) achieve acceptable throughput by never letting the channel go idle if some ready terminal is available. More precisely, a ready terminal senses the channel and operates as follows:

- If the channel is sensed idle, it transmits the packet with probability one.
- If the channel is sensed busy, it waits until the channel goes idle (i.e., *persisting* on transmitting) and only then transmits the packet (with probability one—hence, the name 1-Persistent).

A slotted version of the 1-Persistent CSMA can be considered in which the time axis is slotted and the slot size is τ seconds (the propagation delay). All terminals are synchronized and are forced to start transmission only at the beginning of a slot. When a packet's arrival occurs during a slot, the terminal senses the channel at the beginning of the next slot and operates according to the protocol described above.

Non-Persistent CSMA

While the previous protocol was meant to make "full" use of the channel, the idea here is to limit the interference among packets by always rescheduling a packet which finds the channel busy upon its arrival. On the other hand, this scheme may introduce idle periods between two consecutive non-overlapped transmissions. More precisely, a ready terminal senses the channel and operates as follows:

- If the channel is sensed idle, it transmits the packet.
- If the channel is sensed busy, then the terminal schedules the retransmission of the packet to some later time according to the retransmission delay distribution. At this new point in time, it senses the channel and repeats the algorithm described.

A slotted version of this Non-Persistent CSMA can also be considered by slotting the time axis and synchronizing the transmission of packets in much the same way as for the previous protocol.

p-Persistent CSMA

The two previous protocols differ by the probability (one or zero) of not rescheduling a packet which upon arrival finds the channel busy. In the case of a 1-Persistent CSMA, we note that whenever two or more terminals become ready during a transmission period, they wait for the channel to become idle (at the end of that transmission) and then they all transmit with probability one. A conflict will also occur with probability one! The idea of randomizing the starting times of transmission of packets accumulating at the end of a transmission period suggests itself for interference reduction and throughput improvement. The scheme consists of including an additional parameter p, the probability that a ready packet persists ($1-p$ being the probability of delaying transmission by τ seconds). The parameter p will be chosen so as to reduce the level of interference while keeping the idle periods between any two consecutive non-overlapped transmissions as small as possible.

More precisely, the protocol consists of the following: the time axis is slotted where the slot size is τ seconds. For simplicity of analysis, we consider the system to be synchronized such that all packets begin their transmission at the beginning of a slot.

Consider a ready terminal:

- If the channel is sensed idle, then

  —with probability p, the terminal transmits the packet.
  —with probability $1-p$, the terminal delays the transmission of the packet by τ seconds (i.e., one slot). If at this new point in time, the channel is still detected idle, the same process above is repeated; otherwise, some packet must have started transmission, and our terminal schedules the retransmission of the packet according to the retransmission delay distribution (i.e., acts as if it had conflicted and learned about the conflict).

- If the ready terminal senses the channel busy, it waits until it becomes idle (at the end of the current transmission) and then operates as above.

* By considering this constant propagation delay equal to the largest possible, one gets lower (i.e., pessimistic) bounds on performance.

Note that 1-Persistent is the special case of p-Persistent with $p=1$.

Throughput equations

We assume that our traffic source consists of a very large number M of users who collectively can be approximated by an independent Poisson source with an aggregate mean packet generation rate of λ packets/second. This implies that each user will generate packets infrequently and each packet can be successfully transmitted in a time interval much less than the average time between successive packets generated by a given user.

In addition, we characterize the traffic as follows. We have assumed that each packet is of constant length requiring T seconds for transmission. Let $S=\lambda T$. S is the average number of new packets generated per transmission time, i.e., the input rate normalized with respect to T. If we were able to perfectly schedule the packets into the available channel space with absolutely no overlap or space between the packets, we would have $S=1$; therefore, we also refer to S as the *channel utilization*, or *throughput*. The maximum achievable throughput for an access mode is called the *capacity* of the channel under that mode.

Each user delays the transmission of a previously collided packet by some random time (introduced to avoid repeated conflicts) whose mean is $\bar{X}$ (chosen, for example, uniformly between 0 and $X_{\max}=2\bar{X}$). Since conflicts can occur, the traffic offered to the channel from our collection of users consists of new packets and previously collided packets. This increases the mean offered traffic rate to G packets per transmission time T, where $G \geq S$.

Our two further assumptions are:

(A1) The average retransmission delay $\bar{X}$ is large compared to T.

(A2) The interarrival times of the point process defined by the start times of all the packets plus retransmissions are independent and exponentially distributed.

We wish to solve for the channel capacity of the system for all of the access protocols described above. This we do by expressing S in terms of G (as well as other system parameters). The channel capacity is obtained by maximizing S with respect to G. Note that S/G is merely the probability of a successful transmission and G/S is the average number of times a packet must be transmitted or scheduled until success.

The basic equations for the throughput S are expressed in terms of a (the ratio of propagation delay to packet transmission time) and G (the offered traffic rate) as follows:*

1-Persistent CSMA

$$S=\frac{G[1+G+aG(1+G+aG/2)]e^{-G(1+2a)}}{G(1+2a)-(1-e^{-aG})+(1+aG)e^{-G(1+a)}} \qquad (1)$$

Slotted 1-Persistent CSMA

$$S=\frac{Ge^{-G(1+a)}[1+a-e^{-aG}]}{(1+a)(1-e^{-aG})+ae^{-G(1+a)}} \qquad (2)$$

Non-Persistent CSMA

$$S=\frac{Ge^{-aG}}{G(1+2a)+e^{-aG}} \qquad (3)$$

Slotted Non-Persistent CSMA

$$S=\frac{aGe^{-aG}}{(1+a)(1-e^{-aG})+a} \qquad (4)$$

p-Persistent CSMA

$$S(G,p,a)=\frac{(1-e^{-aG})[P_s'\pi_0+P_s(1-\pi_0)]}{(1-e^{-aG})[a\bar{t}'\pi_0+a\bar{t}(1-\pi_0)+1+a]+a\pi_0} \qquad (5)$$

where P_s', P_s, $\bar{t}'$, $\bar{t}$ and π_0 are defined in Reference 13. We note that

$$S(G, p \to 0, a=0) \to \frac{G}{G+e^{-G}}$$

In Figure 4 for $a=0.01$, we plot S versus G for the various access modes introduced so far and show the relative performance of each. We also summarize these results in the following table:

PROTOCOL	CAPACITY C
Pure ALOHA	0.184
Slotted ALOHA	0.368
1-Persistent CSMA	0.529
Slotted 1-Persistent CSMA	0.531
0.1-Persistent CSMA	0.791
Non-Persistent CSMA	0.815
0.03-Persistent CSMA	0.827
Slotted Non-Persistent CSMA	0.857
Perfect Scheduling	1.000

While the capacity of ALOHA channels does not depend on the propagation delay, the capacity of a CSMA channel does. An increase in a increases the "vulnerable" period of a packet and reduces its capacity. This also results in "older" channel state information from sensing. In Figure 5 we plot, versus a, the channel capacity for all of the above random access modes. For large a, we note that slotted ALOHA (and even "pure" ALOHA) is superior to any CSMA mode since decisions based on partially obsolete data are deleterious; this effect is due in part to our assumption about the constant propagation delay.

* For proofs, the reader is referred to Reference 13.

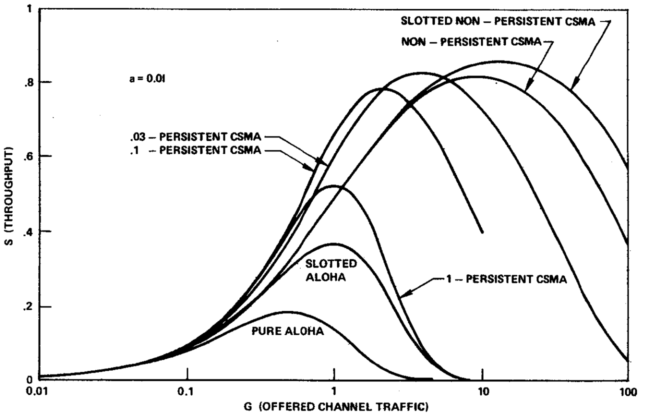


Figure 4—Throughput for the various random access modes ($a=0.01$)

Delay performance

We introduce at this point the expected packet delay $\mathfrak{D}$ defined as the average time from when a packet is generated until it is successfully received. Our principal concern in this section is to investigate the tradeoff between the average delay and the throughput S.

For the present study, it is assumed that the acknowledgment packets are always correctly received with probability one. The simplest way to accomplish this is to create a separate channel to handle acknowledgment traffic. If sufficient bandwidth is provided, overlaps between acknowledgment packets are avoided, since a positive acknowledgment packet is created only when a packet is correctly received, and there will be at most one such packet at any given time. Thus, if T_a denotes the transmission time of the acknowledgment packet on the separate channel, then the time-out for receiving a positive acknowledgment is $T+\tau+T_a+\tau$, provided that the processing time needed to perform the sumcheck and to generate the acknowledgment packet is assumed negligible.

The Delay $\mathfrak{D}$ is a function of S and $\bar{X}$. Thus, for each S, a minimum delay can be achieved by choosing an optimal $\bar{X}$. Such an optimization problem is difficult to solve analytically, and simulation techniques have been employed.

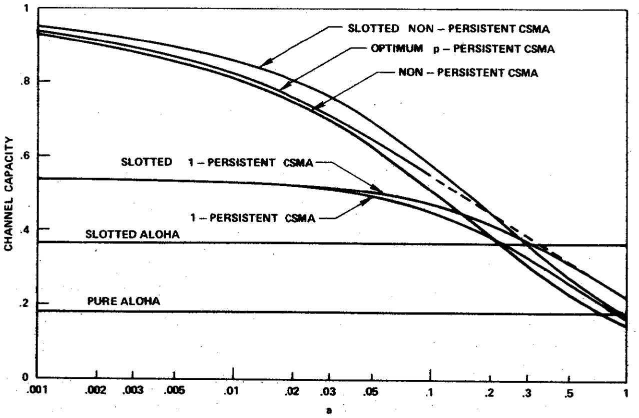


Figure 5—CSMA and ALOHA:
Effect of propagation delay on channel capacity

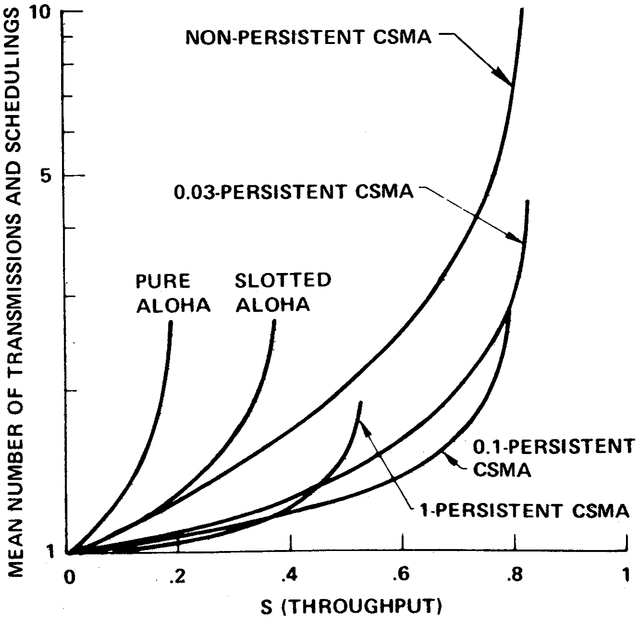


Figure 6—CSMA and ALOHA:
G/S versus throughput ($a=0.01$)

Before we proceed with the discussion of the simulation results, we compare the various access modes in terms of the average number of transmissions (or average number of schedulings) G/S. For this purpose, we plot G/S versus S in Figure 6 for the ALOHA and CSMA modes, when $a=0.01$. Note that CSMA modes provide lower values for G/S than the ALOHA modes. Furthermore, for each value of the throughput, there exists, a value of p such that p-Persistent is optimal. For small values of S, $p=1$ (i.e., 1-Persistent) is optimal. As S increases, the optimal p decreases.

Simulation results

The simulation model is based on all system assumptions presented above. However, we relax the assumptions concerning the retransmission delay and the independence of arrivals for the offered channel traffic.

In general, our simulation results indicate the following:

(1) For each value of the input rate S, there is a minimum value $\bar{X}$ for the average retransmission delay variable, such that below that value, it is impossible to achieve a throughput equal to the input rate. The higher S is, the larger $\bar{X}$ must be to prevent a constantly increasing backlog, i.e., to prevent the channel from saturating. In other words, the maximum achievable throughput (under stable conditions) is a function of $\bar{X}$, and the larger $\bar{X}$ is; the higher is the maximum throughput.

(2) Recall that the throughput equations were based on

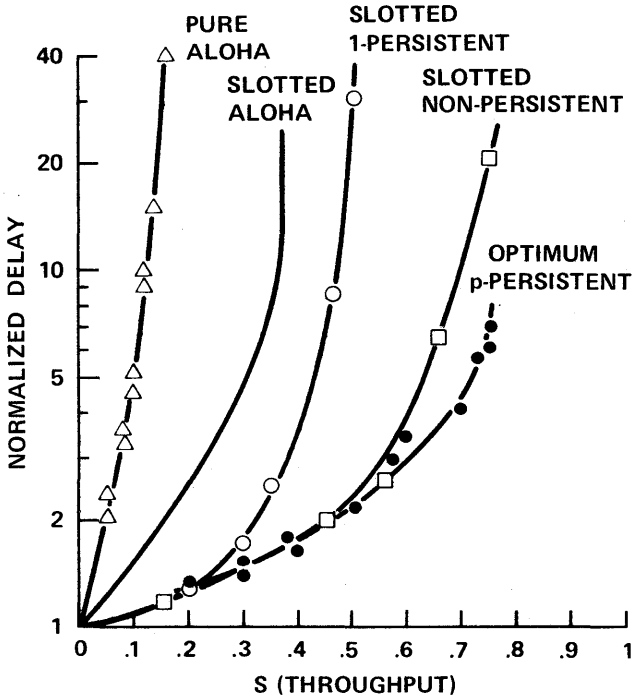


Figure 7—CSMA and ALOHA: Throughput-delay trade-offs from simulation ($a=0.01$)

the assumption that $\bar{X}$ is infinitely large compared to T. Simulation shows that for finite values of $\bar{X}$, larger than some value $\bar{X}_0$ but not too large compared to T, the system already "reaches" the asymptotic results ($\bar{X}\rightarrow\infty$), i.e., for some finite values of $\bar{X}$, assumption (A2) is satisfied and delays are acceptable. Simulation experiments were conducted to find the optimal delay, that is, the value of $\bar{X}(S)$ which allows one to achieve the indicated throughput with the minimum delay.

Finally, in Figure 7, we give the throughput-minimum delay trade-off for the three Carrier Sense Multiple Access modes and $a=0.01$. This is the basic performance curve.

THE EFFECT OF HIDDEN TERMINALS ON CHANNEL CAPACITY FOR CARRIER SENSE MULTIPLE ACCESS

The performance obtained in the previous section (in terms of channel capacity and throughput-delay trade-offs) was based on the (strong) assumption that all terminals were in line-of-sight and within range of each other. There are many instances where this is not the case, forcing us to relax that assumption. Two terminals can be within range of the station but out-of-range of each other, or, they can be separated by some physical obstacle opaque to UHF radio signals. Two such terminals are then said to be "hidden" from each other. It is evident that the existence of hidden elements in an environment affects (degrades) the performance of CSMA. In this section we discuss this effect. (For simplicity, we restrict our study to 1-Persistent and Non-Persistent CSMA protocols only.)

Definitions and representation of configurations with hidden elements

In the sequel, terminals are in line-of-sight and within range of the station, but not necessarily with respect to each other. By definition, terminal i "hears" (is connected to) terminal j if i and j are within range and in line-of-sight of each other. In order to represent terminal configurations with hidden elements, it is advantageous to partition the population into several groups (say N) such that all terminals in a group hear exactly the same subset of terminals in the population. (This partitioning is easily formed if we know the hearing matrix of the population. See References 14 and 15). Let $h(i)$ be the set of groups that group i can hear.

We shall further assume that each group i consists of a large number of users who collectively form an independent Poisson source with an aggregate mean packet generation rate λ_i packets per second such that $\sum_{i=1}^{N}\lambda_i=\lambda$. Let $S_i=\lambda_i T$ and $S=\lambda T=\sum_{i=1}^{N}S_i$; S is the total throughput of the channel.

Let $\mathbf{S}=(S_1, S_2, \ldots, S_N)$.

We can write $\mathbf{S}$ as $\mathbf{S}=S\mathbf{U}$ such that

$$u_i\geq 0\forall i$$

and

$$\|\mathbf{U}\|\triangleq\sum_{i=1}^{N}u_i=1$$

(The vector $\mathbf{U}$ describes a direction in N-dimensional space.) The *capacity* of the channel along the direction $\mathbf{U}$ is defined as

$$C(\mathbf{U})=\underset{0\leq S\leq 1}{\text{Maximum}}\ S$$

such that the set of inputs determined by the vector $\mathbf{S}(\mathbf{U})$ is achievable. Equivalently, we say that a set $\mathbf{S}(\mathbf{U})$ of input rates is feasible if and only if

$$\mathbf{S}(\mathbf{U})\leq C(\mathbf{U})$$

Let G_i denote the mean offered traffic rate of group $i(G_i\geq S_i)$. Let $\mathbf{G}=(G_1, G_2, \ldots, G_N)$ and $G=\sum_{i=1}^{N}G_i$. Finally, we consider $\bar{X}$ to be the same for all groups and the assumptions concerning the retransmission delay and the independence of arrivals for the offered traffic to still hold true.

Throughput equations

We recognize that S_i/G_i is merely the probability of success of an arbitrary packet from group i. This quantity is a function of the traffic vector $\mathbf{G}$. By expressing S_i/G_i for each i in terms of $\mathbf{G}$, we obtain a set of equations relating the components of $\mathbf{S}$ to the components of $\mathbf{G}$.

In the case of *independent* groups (i.e., such that terminals in a group do not hear terminals in other groups) for a given $\mathbf{G}$ and under the system and model assumptions stated above, the probability of success of an arbitrary packet from group i is given as follows*

For 1-Persistent CSMA

$$P_{s_i}=\frac{S_i}{G_i}=\frac{[1+G_i+aG_i(1+G_i+aG_i/2)]}{(1+aG_i)e^{-G_i(1-2a)}}$$

$$\prod_{j=1}^{N}\frac{(1+aG_j)e^{-2G_j}}{G_j(1+2a)-(1-e^{-aG_j})+(1+aG_j)e^{-G_j(1+a)}} \qquad (6)$$

For Non-Persistent CSMA

$$P_{s_i}=\frac{S_i}{G_i}=e^{G_i(1-2a)}\prod_{j=1}^{N}\frac{e^{-G_j(1-a)}}{G_j(1+2a)+e^{-aG_j}} \qquad (7)$$

This set of equations relates the components of the input vector **S** to the components of the traffic vector **G**. For a given input vector **S**, we can numerically solve for G_i, $i=1, \ldots, N$. This we do by writing the above equations in the form

$$G_i=S_i/f_i(G_1, \ldots, G_N)$$

where f_i is a function of the vector **G**, and by solving the set of equations iteratively, starting with the initial values **G** = **S**. If the input vector is a feasible one, then the iterative procedure will result in a (finite) traffic vector **G**, satisfying the above set of equations. Thus the convergence of the iterative procedure determines the feasibility of the input vector **S** and the final values G_i/S_i, $i=1, 2, \ldots, N$ give the average number of transmissions and schedulings a packet from group i undertakes before success. This will be our measure of relative performance of the various groups. Some simple examples are treated in the following section.

In the case of *dependent* groups, similar but approximate relationships can be found for the Non-Persistent CSMA protocol. They are expressed as

$$S_i=G_i\frac{\prod_{j\in h(i)} e^{-aG_j'}\prod_{k\in h(i)} e^{-G_k'(1-a)}}{\prod_{l=1}^{N}[G_l'(1+2a)+e^{-aG_l'}]} \qquad (8)$$

$$G_i'=G_i\prod_{\substack{j\in h(i)\\ j\neq i}}\frac{1+aG_j'}{G_j'(1+2a)+e^{-aG_j'}} \qquad (9)$$

Examples

Here we consider some typical examples of independent groups to which we apply the analytical results found above. Simulation techniques have been used to check the validity of the assumptions on which the analysis was based. We restrict ourselves to $a=0.01$.

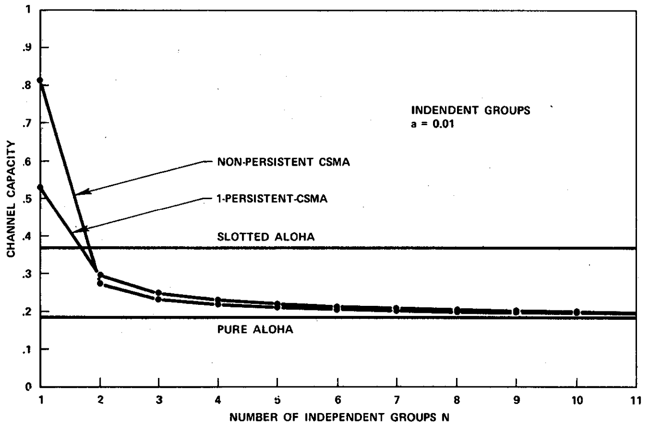


Figure 8—Independent group case: Channel capacity versus the number of groups

Independent Groups Case—A Symmetric Configuration

The population is partitioned into N groups of equal size. For each terminal there exists a fraction β of the population which is hidden, namely $\beta=(N-1)/N(\geq 0.5)$. The channel capacity for various values of N is plotted in Figure 8. Note that the channel capacity experiences a drastic decrease between the two cases: $N=1$ (no hidden terminals, $\beta=0$) and $N=2(\beta=0.5)$. For $N\geq 2$, slotted ALOHA performs better than CSMA. This decrease is more critical for the Non-Persistent CSMA than for the 1-Persistent CSMA as shown in the Figure. For $N>2$, the channel capacity is rather insensitive to N and approaches pure ALOHA for large N.

Independent Groups Case—Complementary Couple Configuration

The previous example did not show the effect of a small fraction of the population being hidden from the rest. In this example the population consists of two independent groups ($N=2$) of unequal sizes such that $\mathbf{U}=(\alpha, 1-\alpha)$ that is

$$S_1=\alpha S$$

$$S_2=(1-\alpha)S$$

Equations (6) and (7) are readily applicable. The channel capacity is plotted versus α for both CSMA protocols in Figure 9. Here again we note that the capacity decreases rapidly as α increases from 0. This decrease is much more critical for the Non-Persistent than for the 1-Persistent. As soon as $\alpha=10^{-2}$, the capacity of Non-Persistent CSMA is only 0.5, as compared to 0.82 when $\alpha=0$. In addition, CSMA performs (capacity-wise) only as good as slotted ALOHA as soon as $\alpha=0.08$ for the Non-Persistent protocol and $\alpha=0.1$ for the 1-Persistent protocol. In both cases, we note that the minimum capacity is obtained for $\alpha=0.5$; this corresponds to the case $N=2$ in the previous example.

* See References 14 and 15 for proof.

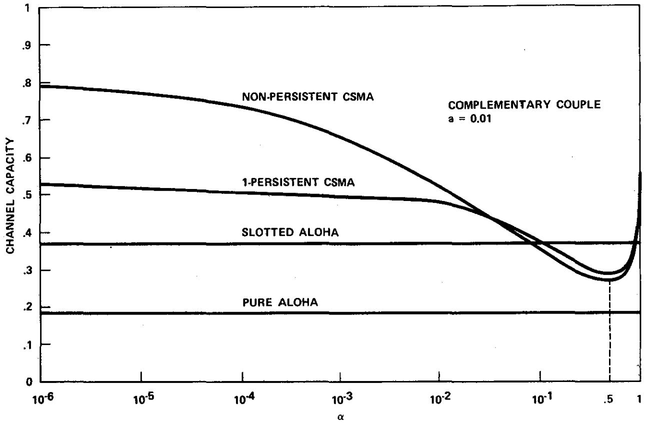


Figure 9—Complementary couple configuration: Channel capacity versus α

In addition, we simulated the 1-Persistent CSMA case for this example and various values of α. The comparison of (S_1, G_1) and (S_2, G_2) relationships obtained from simulation to the results obtained from the analytical model exhibits an excellent match, thus checking the validity of the model.

Examining G_i/S_i for each group, we noted that the large group always performed better than the smaller one. Although we noted for $\alpha \simeq 0.1$ that 1-Persistent CSMA has a capacity only as great as slotted ALOHA, the average number of transmissions $\alpha(G_1/S_1)+[(1-\alpha)(G_2/S_2)]$ was lower (superior) for the 1-Persistent CSMA than for slotted ALOHA.

CARRIER SENSE MULTIPLE ACCESS WITH A BUSY TONE

System operation

In this section we wish to consider a solution to the hidden terminal problem which we call the **B**usy **T**one **M**ultiple **A**ccess mode (BTMA). The operation of BTMA rests on the assumption that the station is, by definition, within range and in line-of-sight of all terminals. The total available bandwidth is to be divided into two channels: a message channel and a busy tone (BT) channel. As long as the station senses a (terminal) carrier on the incoming message channel it transmits a (sine wave) busy tone signal on the busy tone channel. It is by sensing a carrier on the busy tone channel that terminals determine the state of the message channel. The action pertaining to the transmission of the packet that a terminal takes (again) is prescribed by the particular protocol being used. We shall restrict ourselves to the *Non-Persistent protocol* because of its simplicity in analysis and implementation, as well as its relatively high efficiency as shown above. In CSMA, the difficulty of detecting the presence of a signal on the message channel when this message uses the entire bandwidth is minor and therefore is neglected. It is not so when we are concerned with the (statistical) detection of the (sine wave) busy tone signal on a narrow band channel. The detection time, denoted by t_d, is no longer negligible and must be accounted for. The Non-Persistent BTMA protocol is similar to the Non-Persistent CSMA protocol and corresponds to the following. Whenever a terminal has a packet ready for transmission, it senses the busy tone channel for t_d seconds (the detection time) at the end of which it decides whether the BT signal is present or absent. (t_d is a system parameter and its optimal value is discussed below). If the terminal decides that the BT signal is absent then it transmits the packet, otherwise it reschedules the packet for transmission at some later time incurring a random rescheduling delay; at this new point in time, it senses the BT channel and repeats the algorithm. In the event of a conflict, which the terminal learns about by failing to receive an acknowledgment from the station, the terminal again reschedules the transmission of the packet for some later time, and repeats the above process.

Of interest is first, the determination of the channel capacity under a Non-Persistent BTMA protocol and second, the throughput delay characteristics of the latter. The total available bandwidth being the limiting resource, the problem then reduces to selecting the system parameters in order to achieve the best system performance.

Here we make the same assumptions as above. However while the effect of noise is assumed to be negligible on the message channel, we do account for it in the (narrow band) busy tone channel. Each packet is of constant length requiring T_m seconds for transmission on the message channel. Let $S_m = \lambda T_m$. S_m is the average number of new packets generated per transmission time, i.e., this is the input rate normalized with respect to T_m. Under steady state conditions, S_m can also be referred to as the message channel throughput rate and as the *message channel utilization*. Let ψ be the fraction of the bandwidth assigned to the BT channel. Let $S=(1-\psi)S_m$. S is the *overall channel utilization*. The maximum achievable channel utilization is the *capacity* of the channel.

Signal detection

The detection of the busy tone signal is the problem of detecting a signal of known form in the presence of noise. The useful signal is a given function with some unknown parameters, namely, phase and amplitude.† However the observation (detection) time is usually small compared to the "fluctuation time" of these parameters, and the unknown phase and amplitude can be regarded as constant.

The problem of detecting a signal in a background of random noise is a classical statistical problem involving the choice of one hypothesis from two mutually exclusive hypotheses. This has been extensively studied in the litera-

† Because of the mobility of terminals, the signal fluctuates. Thus we assume it to be of unknown amplitude. In the case of fixed terminals, we may idealize the problem to be that of detecting a signal with known amplitude but unknown phase.

ture.[16] The quality of the decision can be characterized by two probabilities:

D Probability of correct detection (in presence of the signal)
F Probability of incorrect detection or false alarm

The detector at the receiver consists of a filter, an integrator and a threshold decision box. Assuming the step response of the busy tone detect filter to be exponential, and considering the same peak power to be used for the busy tone as for the message on the message channel, then the signal-to-noise ratio (SNR) $\mu(t)$ on the busy tone channel at time t is given by

$$\mu(t)=\mu_m\frac{1-\psi}{\psi}\,(1-e^{-2\psi Wt})^2 \qquad (10)$$

where

- μ_m is the SNR of the message on the message channel required for suitable operation (typically $\mu_m=10$)
- ψ is the fraction of bandwidth assigned to the BT channel
- the time constant of the filter exponential rise is taken to be $\frac{1}{2}\psi W$.

Consider now a signal starting at $t=0$ and terminating at $t=T$. Let $D(t)$ be the probability of correct detection at time t after having observed the channel over t_d seconds (t is the time at which the decision is made). $D(t)$ is determined by (See Reference 16).

$$D(t)=F^{(1/1+\mu(u))} \qquad (11)$$

where

$$u=\begin{cases} t & \text{if} \quad 0\leq t\leq t_d \\ t_d & \text{if} \quad t_d\leq t\leq T \\ T-t+t_d & \text{if} \quad T\leq t\leq T+t_d \end{cases}$$

For $t>T+t_d$, the probability of false alarm is F.

Throughput equation

We wish to solve for the channel capacity, given the system parameters F, ψ, W, b_m, τ, t_d. This we do by solving for S in terms of γ (the traffic rate measured in packets per second) and other parameters. The channel capacity is then found by maximizing S with respect to γ.

Contrary to the CSMA modes the fraction of the population which decides to transmit is a function of time. The analytical approach consists of identifying the busy and idle periods and of determining the condition for a successful transmission over the busy period. To keep the analysis simple, some very minor approximations are made yielding a lower bound on throughput as given in the following equation*

$$S\geq S_l=\frac{b_m}{W}\frac{\exp\left[-\gamma m(0,T_m)\right]}{\bar{B}+\bar{I}} \qquad (12)$$

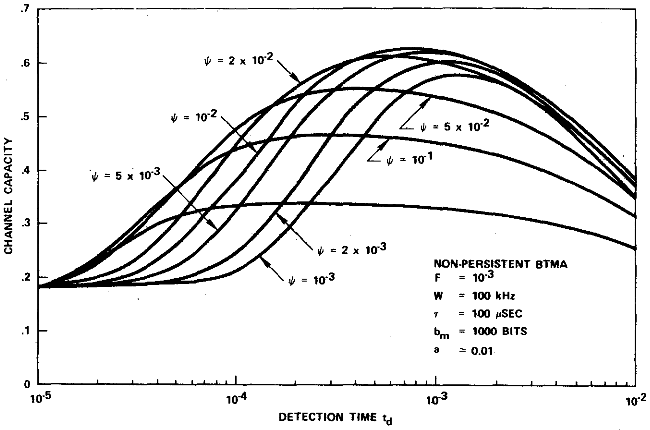


Figure 10—BTMA: Channel capacity versus observation window t_d

It can also be shown that in the limit, when $t_d\rightarrow 0$, the channel capacity reduces to

$$S=(1-\psi)\,\frac{1}{2e} \qquad (13)$$

Results

The design problem in BTMA consists of maximizing the channel capacity (under the Non-Persistent protocol) by properly selecting the design variables ψ, F and t_d when the number of bits per packet, b_m, and the total available bandwidth W are given. Because of the complicated form of the expressions for S, numerical optimization techniques are used.

To reduce the dimensionality of the problem, and to provide an easy comparison with the previously analyzed CSMA protocols we restrict ourselves to the following:

- τ (maximum propagation delay) = 100 μsec.*
- $\mu_m=10$
- $\dfrac{b_m}{W}=10^2$ msec.†

We consider two cases for b_m and W:

case I: $b_m=1000$ bits; $W=10^5$ Hz
case II: $b_m=10{,}000$ bits; $W=10^6$ Hz

* See References 14 and 15 for proof and for the definition of $m(0, T_m)$, $\bar{B}$ and $\bar{I}$.

* The bandwidth is assumed to be modulated a 1 bit/Hz-sec.

† This corresponds to a maximum distance of about 20 miles. The ratio of propagation delay to transmission time of a packet, denoted by a, is, in all cases less than (but very close to) or equal to 0.01.

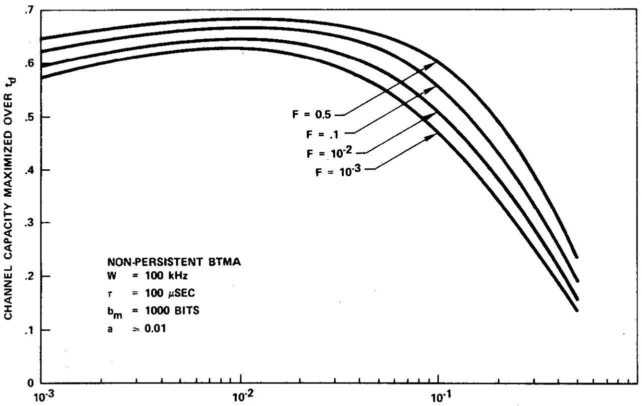


Figure 11—BTMA: Channel capacity (maximized over t_d) versus ψ

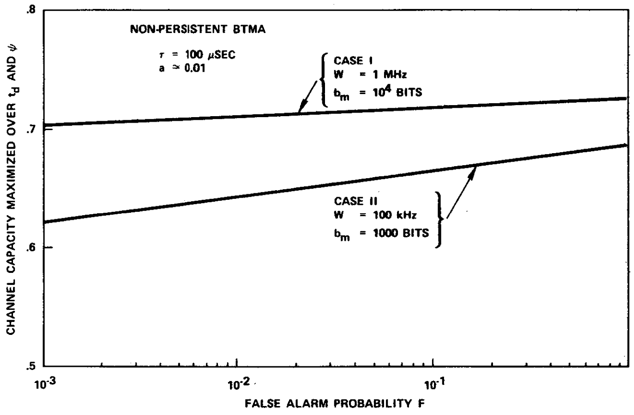


Figure 12—BTMA: Channel capacity (maximized over t_d and ψ) versus F

For $F=10^{-3}$ and various values of ψ we plot in Figure 10 the channel capacity versus the observation window t_d. Similar curves can be plotted for other values of F. For each couple (F, ψ) the channel capacity reaches its maximum at some optimum value of t_d. This optimum is explained by the fact that the larger t_d is, the better is the probability of correct detection $D(t_d)$ when the signal is present during the entire window. However, the larger t_d is, the longer the idle period will be. The effect is reversed as t_d gets smaller.

Note that when the observation window shrinks to 0, the capacity of the channel decreases to $(1-\psi)\frac{1}{2}e$, the capacity provided by the pure ALOHA access mode. Qualitatively speaking $t_d \rightarrow 0$ reduces to very bad detection, and terminals behave in a pure ALOHA mode.

In Figure 11, we plot for various F, the maximum capacity of the channel (maximized over t_d) versus ψ. We note here that the maximum capacity is not very sensitive to *small* variations of ψ˙ However, there is a certain range of ψ which yields the best performance. For those values of F considered in the graph ($F=10^{-3}$, 10^{-2}, 10^{-1}, 0.5), the optimum ψ is the range $(10^{-2}, 2\times10^{-2})$.

In Figure 12, we plot the capacity (maximized over ψ and t_d) versus F for cases I and II. Note that for both cases the capacity of the channel is a logarithmic function of F. The ultimate performance (≃0.68 for Case I and ≃0.72 for Case II) is obtained for $F \rightarrow 1$. However, the channel capacity is not very sensitive to variations of F. Case II offers a channel capacity higher than that offered by Case I; we note that this gain does not consider other factors such as increased power requirements.*

To compare the delay performance of BTMA for various values of the system parameters, we first consider the quantity G/S, the average number of transmissions and schedulings that a packet incurs before successful transmission. In Figure 13, we plot, for each value of F, G/S versus S for those values of ψ and t_d yielding the maximum channel capacity. Note that for each value of S there exists a value of F minimizing G/S. However, for relatively small values of S (not too close to the saturation point of the channel) we note that the higher the probability of false alarm F is, the larger is G/S. An explanation can be given by the following fact:

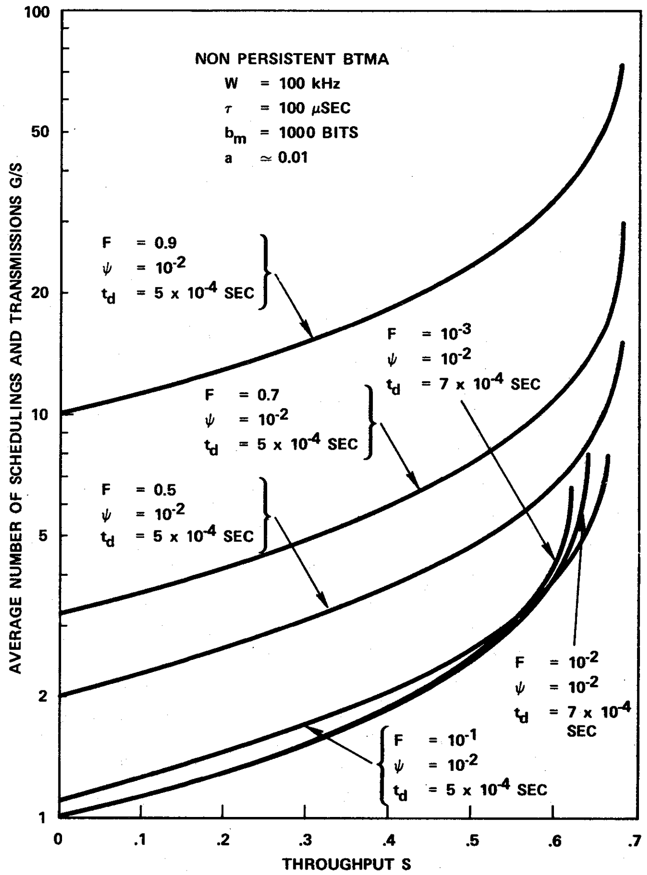


Figure 13—BTMA: Average number of schedulings and transmissions

* The larger the bandwidth is, the better is the correct detection. Thus larger W provides larger channel capacity. However, the channel capacity is always bounded from above by the capacity of CSMA with propagation delay equal to 2τ.[14]

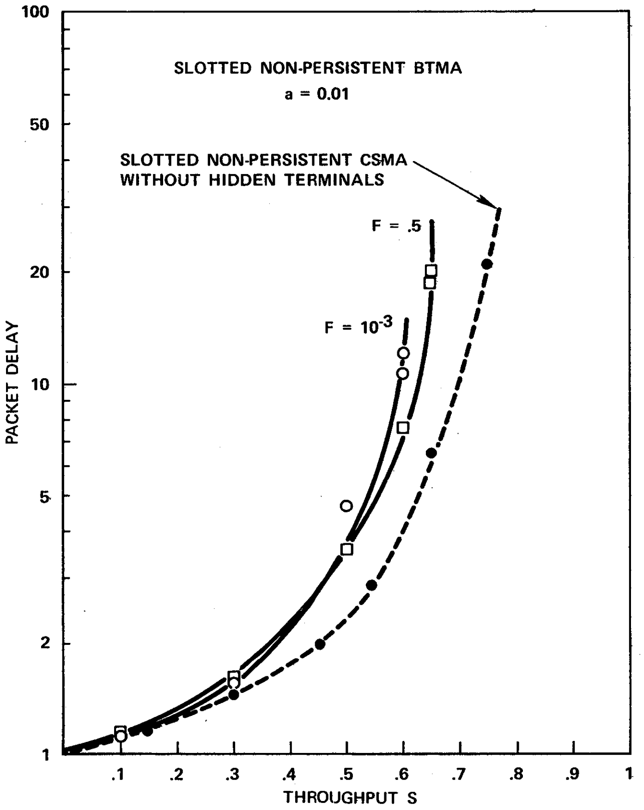


Figure 14—BTMA: Throughput-delay trade-offs ($a=0.01$)

when $G \rightarrow 0$ and $S \rightarrow 0$, the terminal incurs an average number of schedulings and transmissions equal to $1/(1-F)$. This is shown on Figure 13 at $S=0$.

G/S, as a measure of delay, can be of importance since the complexity of the equipment and the implementation of the protocol can be directly related to the number of schedulings and transmissions that a packet incurs. For example, at each scheduling, the terminal has to generate a random number determining the scheduling delay. Of even more importance in evaluating the performance of such a system is the determination of the actual packet delay, defined as the time lapse since the packet is first generated, until the time it is successful. As discussed earlier, the mathematical determination of packet delays is fairly complex, and simulation techniques are employed. For various values of F ($F=10^{-3}$ and $F=0.5$), by selecting the optimum system parameters (ψ, t_d) with respect to channel capacity, we simulated the BTMA mode. In Figure 14 we plot the throughput-minimum-delay* curve for these values of F. It is to be noted that, even though G/S can be significantly affected by F, the minimum delay is relatively insensitive to F. However, for each value of S there exists a value of F which provides the lowest delay. By comparing the lower envelope of these throughput-delay curves to the curve corresponding to the Non-Persistent CSMA without hidden terminals, we note the relatively-good performance of BTMA.

RESERVATION TECHNIQUES

We have shown that, in the presence of a large population of users exhibiting a bursty behavior, FDMA and TDMA produce much higher delays with the same available bandwidth than random multiple access, and in order tc achieve the same delay performance, they require a much larger bandwidth; in the latter case, the utilization of the channel is extremely low. In order to increase the channel utilization beyond FDMA and TDMA, statistical multiplexing or Asynchronous Time Division Multiple Access (ATDMA) has been proposed.[17] However, this technique is less attractive in situations where the terminals are geographically spread and/or mobile.

Of more recent interest are "controlled" techniques for transmission from terminals to computer. There are two methods in common usage for wired networks: contention and polling. In a *contention network,* the terminal makes a request to transmit: if the channel is free, transmission goes ahead; if it is not free, the terminal must wait; the station schedules the transmissions either in a prearranged sequence (according to some scheduling scheme) or in the sequence in which the requests were made. In the *polling technique,* the station asks the terminals one by one whether they have anything to transmit. For this, the station may have a polling list giving the order in which terminals are polled. A polling message is sent to the terminal under consideration. If the terminal has some data to transmit, it goes ahead; if not, a negative reply (or absence of reply) is received, and the next terminal is polled.

These controlled techniques are readily applicable to radio networks. They constitute the subject of this section. It has been shown that although polling may allow the system to achieve high utilization of the channel, the delay incurred by a packet is large (mainly for the large M case which is of interest to us) rendering the polling technique less attractive than CSMA and BTMA. The alternative is the use of reservation techniques. In this section, we study the **S**plit-channel **R**eservation **M**ultiple **A**ccess (SRMA) as one implementation of such reservation techniques. The available bandwidth is divided into two parts: one used to transmit control information, the second used for the message itself.

System operation

In the particular scheme considered here, the bandwidth allocated for control is further divided into two channels:

—the request channel
—the answer-to-request channel.

The request channel will be operated in a random access mode (ALOHA or CSMA). Consider a terminal with a message ready for transmission. To initiate the sending of

* Delay is minimized with respect to $\bar{X}$. In BTMA, the larger F is, the larger is G/S. The minimum delay is obtained for very small values of $\bar{X}$ since the packet incurs $1/(1-F)$ reschedulings when the channel is idle.

the message, the terminal sends, on the request channel, a request packet containing information about the address of the terminal and, in the case of variable length or multipacket messages, the length of the message. At the correct reception of the request packet, the scheduling station computes the time at which the message channel will be available and transmits back to the terminal, on the answer-to-request channel, an answer packet containing the address of the terminal and the time at which it can start transmission.

Analysis

The total delay is composed of the two following components:

(i) $\mathfrak{D}_1$, the time for the request packet to be successfully received at the station, and
(ii) $\mathfrak{D}_2$, the time between reception of the request packet at the station and the end of the message transmission.

Let W_m be the bandwidth allocated to the message channel and $\theta = W_m/W$. The answer-to-request channel is an interference-free channel since the station is the only transmitter. That is, answer packets can be queued at the station and transmitted without conflicts. It is possible to give the answer-to-request channel enough bandwidth W_a such that answer packets do not incur any queueing delay at the station. Indeed, if b_r and b_a are the number of bits per request packet and answer-to-request packet respectively, then W_a should satisfy

$$W_a \geq W_r \frac{b_a}{b_r} \tag{14}$$

where W_r is the bandwidth assigned to the request channel.

Let λ be the average number of messages generated per second. As usual, we shall assume the generation process to be Poisson. The maximum generation rate that the total bandwidth W can ever handle is W/b_m. The channel utilization denoted again by S is then expressed as

$$S = \lambda/(W/b_m) \tag{15}$$

Since both control packets contain the same type of information, it is reasonable to assume that $b_a = b_r$ and therefore let $\eta = b_r/b_m$. We further let $W_r = W_a$. In this case we have

$$W_r = W_a = \frac{(1-\theta)W}{2} \tag{16}$$

Consider the request channel operated in a random access mode. The expected delay incurred by a request packet is readily obtained from the simulation results presented earlier.

To estimate the delay $\mathfrak{D}_2$, we assume that the output of the random access request channel defined as the process corresponding to the arrival of successful requests at the station is Poisson with mean λ requests per seconds. We verified the above assumption by examining the distribution of interdeparture times (i.e., time between successive successful packets) of the Non-Persistent CSMA simulator and comparing it to the exponential density function. Except for interarrivals in the range of one or two packet transmission times the match is acceptable and the smaller S_r is, the more valid is the assumption. Under this assumption, the message channel can be modeled as an $M/G/1$ queueing system.[12]

The *maximum bandwidth utilization* is determined by the fact that the throughput on the request channel does not exceed its capacity (under the access mode in use) and the utilization of the message channel does not exceed one.

Numerical results

System Capacity

In Figure 15 we plot system capacity versus η (which represents a relative measure of the overhead due to control information) for the following access modes:

Pure ALOHA SRMA
Slotted ALOHA SRMA
Slotted Non-Persistent Carrier Sense SRMA
($\tau W/b_m = 0.01$, 0.05)

We note that the system capacity in SRMA reaches 1 for very small η. A case of interest considered throughout the paper corresponds to $b_m = 1000$ bits and b_r anywhere from 10 to 100 bits (b_r is directly related to the number of terminals in the population, since addressing information increases with increasing M). Thus, the interesting range for η is 0.01 to 0.1. For $\eta > 0.01$, the effect on the system capacity of the random access used to operate the request channel is important: a large improvement is gained when the request channel is operated in slotted Non-Persistent CSMA as compared to ALOHA. On the other hand, in comparing the capacity of SRMA to the capacity of random access modes, we note that SRMA can be superior only for relatively small values of η.

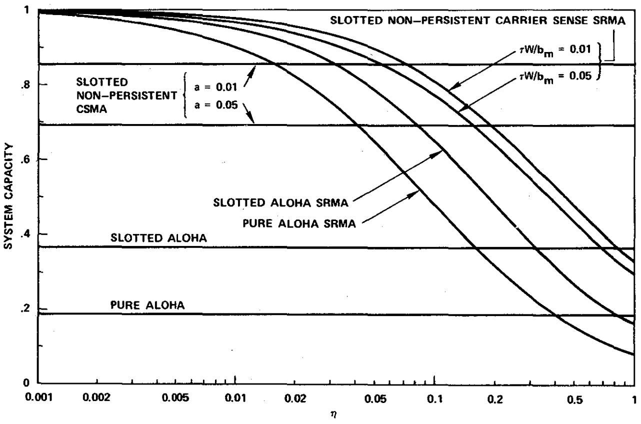


Figure 15—SRMA: Channel capacity versus η

CONCLUSION

Of interest to this paper was the consideration of packet-switched radio channels as a means of communication between terminals and a station (computer center, gate to a network, etc.). The objective of the research was to provide the communication system designer with various new access modes to the shared packet-switched radio channel, as well as the tools and conclusions necessary to select optimal solutions. Carrier Sense Multiple Access (CSMA) was introduced as a new method of multiplexing the terminals on the radio channel. Its performance was shown to be heavily affected by the ratio, a, of propagation delay to packet transmission time. In the cases of interest ($a \ll 1$), and under the major assumption that all terminals are in line-of-sight and within range, we have shown that CSMA provides improved capacity over the ALOHA modes.

However, the existence of hidden terminals can badly degrade the performance of CSMA. A good solution to the problem is provided by the Busy Tone Multiple Access (BTMA). BTMA under a Non-Persistent protocol is shown to achieve a channel capacity of 0.68 when the available bandwidth W is 100 KHz and up to 0.72 when $W=1$ MHz. Moreover, the channel capacity is shown to be insensitive to the precise setting of the system parameters.

A second alternative of multiplexing the terminals on the radio channel is the use of reservation techniques. The

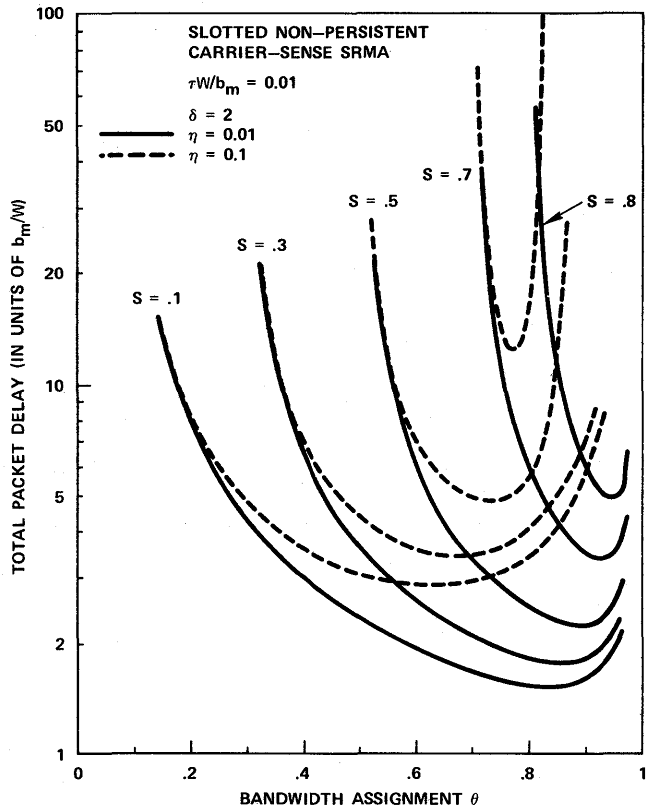


Figure 16—Slotted non-persistent carrier sense SRMA: Packet delay versus bandwith assignment

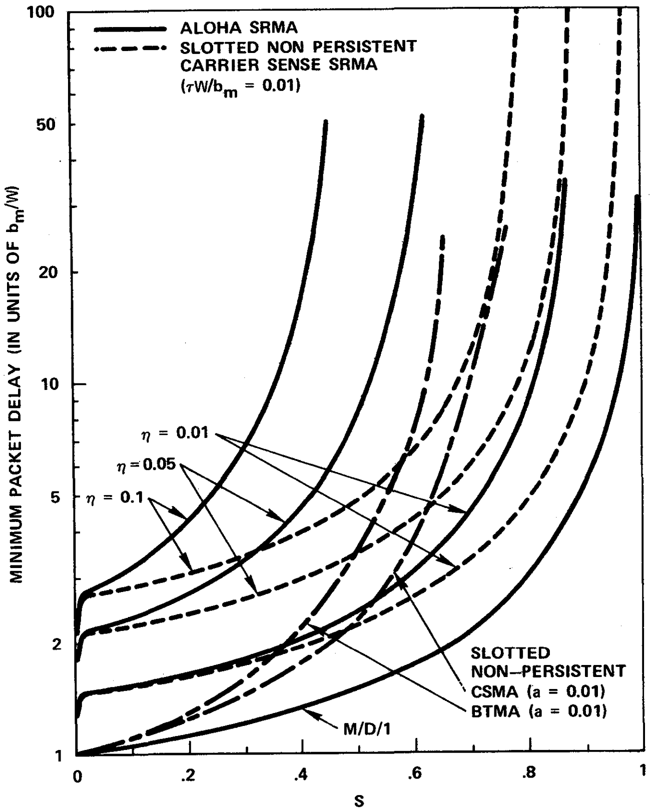


Figure 17—Minimum packet delay in SRMA

Delay Considerations

Let us restrict ourselves here to $\tau W/b_m = 0.01$. For given η and S, the total message delay $\mathfrak{D}$ is a function of θ, the bandwidth assignment. As an example, we show slotted Non-Persistent Carrier Sense SRMA with fixed message length (packet) and $\tau W/b_m = 0.01$ in Figure 16. Similar plots can be obtained for other random access modes used for the request channel. For each value of S, θ must lie in a feasible range denoted as $[\theta_{\min}, \theta_{\max}]$; $\theta_{\min}$ and $\theta_{\max}$ are determined by the saturation of the message channel and the request channel respectively. For small values of θ (θ close to $\theta_{\min}$), the major part of delay is due to $\mathfrak{D}_2$; for θ close to $\theta_{\max}$, it is due to $\mathfrak{D}_1$. The optimal bandwidth assignment is defined as the value of θ which minimizes total delay. We note that the higher the load is, the more critical is the choice of θ_{opt}. The minimum delay for ALOHA-SRMA and Slotted Non-Persistent Carrier Sense SRMA is shown in Figure 17 as a function of S for various values of η. First, in comparing the two systems between themselves, we note again an important improvement in using CSMA for the request channel. The improvement is more important when larger values of η are involved.

In comparing Carrier Sense SRMA with CSMA or BTMA, we note that, unless η is large (0.1 and above), there is a value of S below which CSMA or BTMA performs better than SRMA and above which the opposite is true.

Split-Channel Reservation Multiple Access (SRMA) was considered which employs random access techniques for the request channel. The capacity of the channel under SRMA is heavily affected by the level of overhead introduced. Moreover, the throughput delay performance is significantly dependent on the performance of the random access mode used on the request channel: a Non-Persistent Carrier Sense SRMA provides better performance than ALOHA-SRMA.

In all these comparisons we note that most of the channel capacity which was unavailable with pure and slotted ALOHA may be recovered by use of these more sophisticated access schemes.

REFERENCES

1. Kahn, R. E., "The Organization of Computer Resources into a Packet Radio Network," *NCC 1975 Proceedings.*
2. Abramson, N., R. Binder, F. Kuo, A. Okinaka, and D. Wax, "ALOHA Packet Broadcasting—A Retrospect," *NCC 1975 Proceedings.*
3. Garrett, J. and S. C. Fralick, "A Technology for Packet Radio," *NCC 1975 Proceedings.*
4. Frank, H., R. Van Slyke, and I. Gitman, "Packet Radio Network Design—System Considerations," *NCC 1975 Proceedings.*
5. Fralick, S. C., D. Brandin, F. Kuo, and C. Harrison, "Digital Portable Terminals," *NCC 1975 Proceedings.*
6. Burchfiel, J., R. Tomlinson and M. Beeler, "Functions and Structure of a Packet Radio Station," *NCC 1975 Proceedings.*
7. Abramson, N., "THE ALOHA SYSTEM—Another Alternative for Computer Communications," Fall Joint Computer Conference, *AFIPS Conference Proceedings,* 1970, Vol. 37, pp. 281-285.
8. Kleinrock, L. and S. S. Lam, "Packet-Switching in a Slotted Satellite Channel," National Computer Conference, New York, June 4-8, 1973, AFIPS Conference Proceedings, 1973, Vol. 42, pp. 703-710.
9. Abramson, N., "Packet Switching with Satellites," National Computer Conference, New York, June 4-8, 1973, *AFIPS Conference Proceedings,* 1973, Vol. 42, pp. 695-702.
10. Roberts, L. G., "Dynamic Allocation of Satellite Capacity Through Packet Reservation," National Computer Conference, New York, June 4-8, 1973, *AFIPS Conference Proceedings,* 1973, Vol. 42, pp. 711-716.
11. Spring Joint Computer Conference, *AFIPS Conference Proceedings,* 1970, Vol. 36, pp. 543-597; 1972, Vol. 40, pp. 243-298.
12. Kleinrock, L., *Queueing Systems, Vol. I, Theory, Vol. II, Computer Applications,* Wiley Interscience (1975).
13. Kleinrock, L. and F. A. Tobagi, "Packet Switching in Radio Channels: Carrier Sense Multiple Access Modes and their Throughput-Delay Characteristics," to appear in *IEEE Transactions on Communications.*
14. Tobagi, F. A. and L. Kleinrock, "Packet Switching in Radio Channels: The Hidden Terminal Problem and the Carrier Sense Multiple Access Mode with a Busy Tone," to appear in *IEEE Transactions on Communications.*
15. Tobagi, F. A., *Random Access Techniques for Data Transmission Over Packet Switched Radio Networks,* School of Engineering and Applied Science, University of California, Los Angeles, UCLA-ENGR 7499, December 1974.
16. Wainstein, L. A. and V. D. Zubakov, *Extraction of Signals from Noise,* Prentice Hall, Englewood Cliffs, New Jersey, 1962.
17. Chu, W. W., "A Study of Asynchronous Time Division Multiplexing for Time-Sharing Computer Systems," Spring Joint Computer Conference, *AFIPS Conference Proceedings,* 1969, Vol. 35, pp. 669-678.

ALOHA packet broadcasting—A retrospect

by R. BINDER,* N. ABRAMSON, F. KUO, A. OKINAKA and D. WAX
*University of Hawaii, THE ALOHA SYSTEM***
Honolulu, Hawaii

INTRODUCTION

Packet broadcasting is a technique whereby data is sent from one node in a net to another by attaching address information to the data to form a packet—typically from 30 to 1000 bits in length. The packet is then *broadcast* over a communication channel which is shared by a large number of nodes in the net; as the packet is received by these nodes the address is scanned and the packet is accepted by the proper addressee (or addressees) and ignored by the others. The physical communication channel employed by a packet broadcasting net can be a ground based radio channel, a satellite transponder or a cable.

Packet broadcasting networks can achieve the same efficiencies as packet switched networks,[1] but in addition they have special advantages for local distribution data networks[2] and for data networks using satellite channels.[3] In this paper we concentrate on those characteristics which are of interest for a local distribution data network. In particular, we discuss the lessons learned in the design and implementation of the ALOHANET, a packet broadcasting radio network in operation at the University of Hawaii since 1970. A number of design issues which arose in the construction of the system are defined, our solutions are explained, and in some cases they are justified. The lessons learned from the ALOHANET are used to indicate how such a radio packet broadcasting system might best be built using the technology available in 1975.

In the next section a brief description of the ALOHANET and its rationale is given. This is followed by a detailed discussion of the major system protocol choices that have evolved, pointing out some related theoretical work where appropriate. Choices concerning the design of the radio communication subsystem are then examined, followed by an evolutionary view of the important impact microcomputer technology has had on the user interface design and resulting system capabilities. The concluding section summarizes our present views with respect to the basic system configuration and properties of packet broadcasting nets.

THE ALOHANET

The ALOHANET is the first system which successfully utilized the packet broadcasting concept for on-line access of a central computer via radio. Its primary purpose is to provide inexpensive access to one or more time-sharing systems by a large number of terminal users, typically in the hundreds. However, it also allows user-to-user communication within the net and is evolving toward use in a more generally-oriented computer communications environment.

Operation

The present network configuration makes use of a broadcast channel for only one direction of traffic flow. (As we shall see in later sections, the lack of a broadcast capability in the other direction has seriously handicapped the development of effective protocols in certain areas.) Two 100 KHz channels are used in the UHF band—a *random access* channel for user-to-computer communication at 407.350 MHz and a *broadcast channel* at 413.475 MHz for computer-to-user messages. The original system was configured as a star network, allowing only a central node to receive transmissions in the random access channel; all users received each transmission made by the central node in the broadcast channel. Recently the addition of ALOHA repeaters has generalized the network structure.

A block diagram of the present operational ALOHANET is shown in Figure 1. The central communications processor of the net is an HP 2100 minicomputer (32K of core, 16 bit words) called the MENEHUNE[4] (Hawaiian for IMP) which functions as a message multiplexor/concentrator in much the same way as an ARPANET IMP.[5] The MENEHUNE accepts messages from the UH central computer, an IBM System 360/65 running TSO (as of December 1974, a 370/158) or from ALOHA's own time-sharing computer, the BCC 500, or from any ARPANET computer linked to the MENEHUNE via the ALOHA TIP.[6] Outgoing messages in the MENEHUNE are converted into packets, the packets are queued on a first-in, first-out basis, and are then broadcast to the remote users at a data rate of 9600 baud.

The packet consists of a header (32 bits) and a header parity check word (16 bits), followed by up to 80 bytes of

* Now with Bolt Beranek and Newman, Inc., Cambridge, Massachusetts.

** Supported by the Advanced Research Projects Agency of the Department of Defense and monitored by NASA Ames Research Center under Contract No. NAS2-8590.

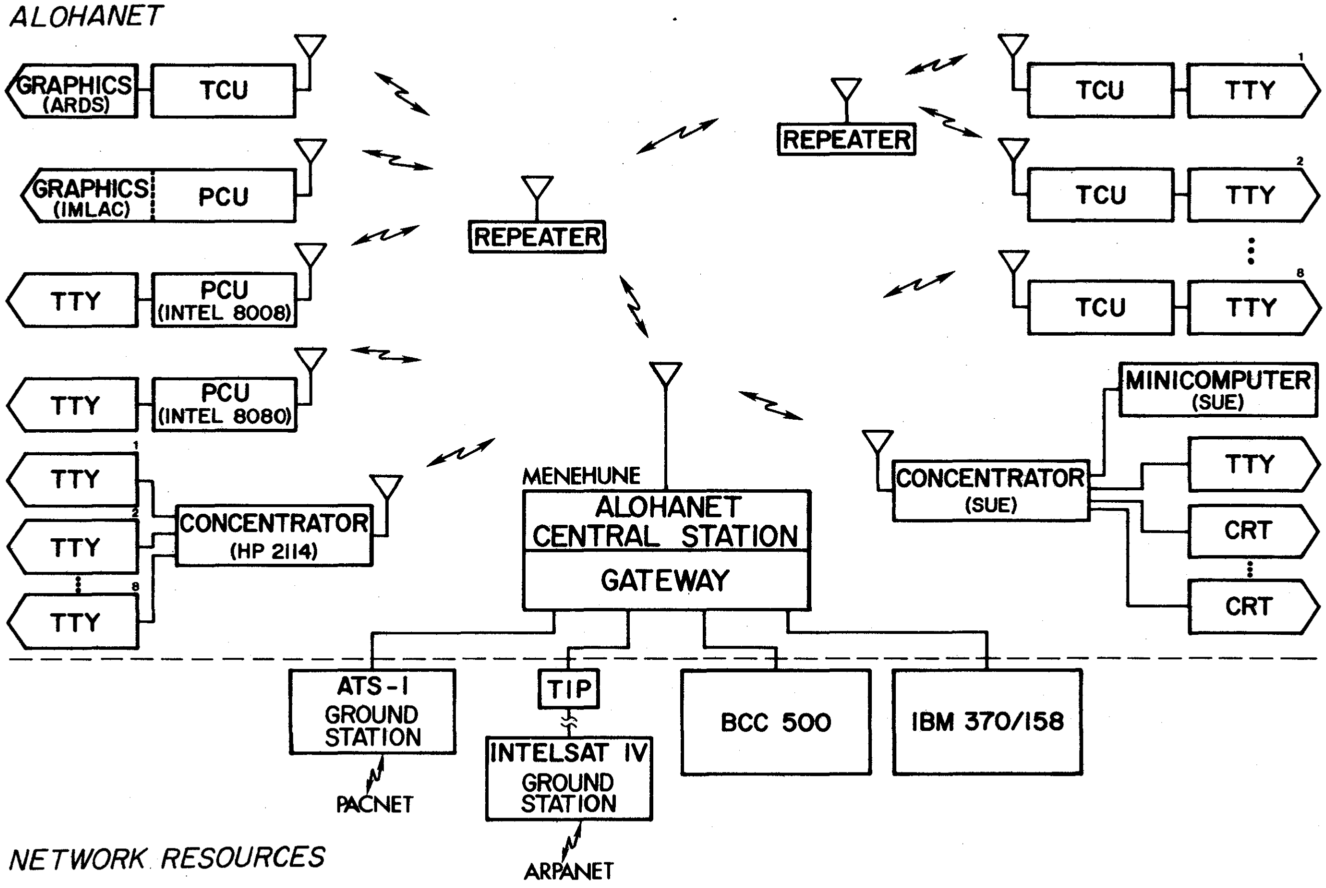


Figure 1—The ALOHANET

data and a 16-bit data parity check word. The header contains information identifying the particular user so that when the MENEHUNE broadcasts a packet, only the intended user's node will accept it. More will be said about packet formats later.

The random access channel (at 407.35 MHz) for communication between users and the MENEHUNE is designed specifically for the traffic characteristics of interactive computing. In a conventional communication system a user might be assigned a portion of the channel on either an FDMA or TDMA basis. Since it is well known that in time-sharing systems, computer and user data streams are bursty,[7] such fixed assignments are generally wasteful of bandwidth because of the high peak-to-average data rates that characterize the traffic. The multiplexing technique that is utilized by the ALOHANET is a purely random access packet switching method that has come to be known as the *pure ALOHA* technique.[8] Under a pure ALOHA mode of operation, packets are sent by the user nodes to the MENEHUNE in a completely unsynchronized manner—when a node is idle it uses none of the channel. Each full packet of 704 bits requires only 73 msecs at a rate of 9600 baud to transmit (neglecting propagation time).

The random or multi-access channel can be regarded as a resource which is shared among a large number of users in much the same way as a multiprocessor's memory is "shared". Each active user node is in contention with all other active users for the user of the MENEHUNE receiver. If two nodes transmit packets at the same time, a collision occurs and both packets are rejected. In the ALOHANET, a positive acknowledgment protocol is used for packets sent on the random-access channel. Whenever a node sends a packet it must receive an acknowledgment message (ACK) from the MENEHUNE within a certain time-out period. If the ACK is not received within this interval the node automatically retransmits the packet after a randomized delay to avoid further collisions. These collisions will limit the number of users and the amount of data which can be transmitted over the channel as loading is increased.

An analysis[8] of the random access method of transmitting packets in a pure ALOHA channel shows that the normalized theoretical capacity of such a channel is $\frac{1}{2}e = 0.184$. Thus the average data rate which can be supported is about one sixth the data rate which could be supported if we were able to synchronize the packets from each user in order to fill up the channel completely. Put another way, this result shows the present 9600 bit/second channel could support between 100 and 500 active teletype users—depending upon the rate at which they generate packets and upon the packet lengths.

ALOHANET remote units

The original user interface developed for the system is an all-hardware unit called an ALOHANET Terminal Control Unit (TCU), and is the sole piece of equipment necessary to connect any terminal or minicomputer into the ALOHA channel. As such it takes the place of two dedicated modems for each user, a dial-up connection and a multiplexor port usually used for computer networks. The TCU is composed of a UHF antenna, transceiver, modem, buffer and control unit.

The buffer and control unit functions of the TCU can also be handled by a minicomputer or a microcomputer. In the present system several minicomputers have been connected in this manner in order to act as multiplexors for terminal clusters or as computing stations with network access for resource sharing. A new version of the TCU using an Intel 8080 microcomputer for buffer and control has been built. Since these programmable units allow a high degree of flexibility for packet formats and system protocols, they are referred to as PCU's (Programmable Control Unit). A more detailed discussion of terminal considerations is given in a companion paper in these proceedings.[9]

Since the transmission scheme of the ALOHANET is by line-of-sight, the radio range of the transceivers is severely limited by the diversity of terrain (mountains, high rise buildings, heavy foliage) that exists in Hawaii. A recent development has allowed the system to expand its geographical coverage beyond the range of its central transmitting station. Because of the burst nature of the transmissions in the ALOHA channel it is possible to build a simple store-and-forward repeater which accepts a packet within a certain range of ID's and then repeats the packet on the same frequency. Each repeater performs identically and independently for packets directed either to or from the MENEHUNE. Two of the repeaters have been built which extend coverage of the ALOHANET from the island of Oahu to other islands in the Hawaiian chain. These repeaters are discussed in more detail in the following section.

PROTOCOL CHOICES

Two fundamental choices which have dictated much of the system protocol are the two-channel star configuration of the original network and the use of random accessing for user transmissions. Investigation of the random accessing principle using radio was in fact the original motivation for constructing the ALOHANET, while the two-channel configuration was primarily chosen to allow this investigation without complication from the relatively dense total traffic stream being returned to all users. An additional reason for the star configuration was the desire to centralize as many communication functions as possible at the MENEHUNE, minimizing the cost of the TCU at each user node.

Within this context, a number of protocol issues must be resolved. The more important of these are:

- random access channel control
- broadcast channel queueing
- packet length
- addressing
- error control
- flow control

Many of the original choices in these areas have undergone significant changes as a result of new user resources and user interfaces, or in some instances due to advancements in theoretical knowledge. The addition of repeaters has (potentially) a particularly significant impact on protocol.

We now discuss some of the considerations and resulting choices made in each of the above areas, with the impacts of new factors introduced within the context of each area. The section concludes with a brief discussion of the problem of integrating file traffic into the random access channel, a subject of current concern in the ALOHANET.

Random access channel control

The retransmission strategy used in the random access scheme plays a central role in the scheme's effectiveness. Its determination directly affects the average delay experienced by users for a successful transmission, given a certain number of users accessing the channel, their traffic statistics, and the channel capacity. It can also be used to prevent the occurrence of channel saturation, a situation in which the channel becomes filled with retransmissions and the number of successful packets falls to zero. These topics have only recently been quantified[10,11] and remain subjects of current investigation.

One approach is to use different constant retransmission intervals at each node, with the intervals equal to integer multiples of the maximum packet transmission time to avoid subsequent conflicts. This results in a priority structure, since nodes assigned the longer intervals will experience a correspondingly longer average delay. As the number of nodes becomes large, however, unacceptably large delays result for the majority of users.

A strategy more appropriate for large user populations is to randomize the retransmission intervals used at each node (note that a priority structure can still be introduced if desired by using larger mean values for lower priority users—in the remaining discussion, equal priorities will be assumed). According to recent results by Lam,[11] the resulting channel behavior appears to be relatively insensitive to the exact nature of the randomization, at least when comparing the use of uniform and geometric distributions. In any event, the cost of implementing a particular distribution at each node is an important design consideration.

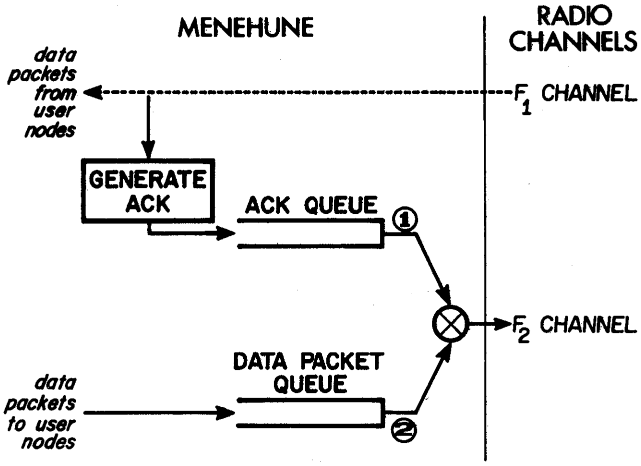


Figure 2—Broadcast Channel Multiplexing

Based on initial estimates of the expected ALOHANET characteristics, a choice was made to use a uniform distribution. This allowed a relatively simple implementation in both hardware and software user nodes.

A simple technique was used in the original system nodes to achieve short delays when the channel is lightly loaded, while preventing channel saturation from occurring due to peak-hour loading or statistical traffic fluctuations: small retransmission intervals are used (relative to the intervals between new packets), but only for a maximum of three successive retransmission attempts. If the third attempt is unsuccessful, the user is notified of a failure and must manually reinitiate the retransmissions. This in effect introduces a long interval between every three retransmissions, allowing time for retransmissions from other users to succeed. Based on a maximum packet transmission time of 70 milliseconds, the intervals are selected from a range of 0.2 to 1.5 seconds, giving a mean of about 0.7 seconds (ten maximum packet times) per retransmission. The lower bound is chosen to allow sufficient time to receive an ACK from the MENEHUNE if the packet was sent successfully, avoiding unnecessary retransmissions. (This time is based on a direct user-MENEHUNE path; if repeaters form a part of the radio path, the lower limit must be increased accordingly.)

The newer programmable PCU's in the system offer the capability of a more flexible strategy, for example allowing the interval used after each third retransmission to be automatically inserted. The use of different strategies, such as continuously increasing the time range used for selection of successive retransmissions, is also easily implemented by program; these and other strategies are currently under investigation.

Broadcast channel queueing

The MENEHUNE acts as a concentrator for the broadcast (F_2) channel, queueing waiting traffic when necessary for sequential transmission to user nodes. Four complicating factors exist, however: a need for priority queueing, fair allocation of the channel, the turnaround delay required by half duplex nodes, and the presence of repeaters.

Priority queues

It is important that the F_2 channel data traffic not prevent the prompt return of an ACK to a user node, since this could lead to unnecessary user retransmissions and possible degradation of the random access (F_1) channel. Thus, an integral part of the F_2 channel multiplexing is the priority queueing mechanism maintained by the MENEHUNE, as shown in Figure 2. Whenever a transmission is completed on the F_2 channel the ACK queue is checked, and if not empty the ACK at the head of the queue is sent. Only when the ACK queue is empty is the data packet queue checked for waiting packets. This guarantees that at most one complete data packet plus any previously queued ACK's will be sent ahead of an ACK just placed on the queue. (Because the average rate of successful arrivals on the F_1 channel is limited to one-sixth the rate of F_2 transmissions by the random access technique, the number of previously queued ACK's will be zero most of the time.)

Fairness

A second problem is the possible hogging of the F_2 channel by one or a few users. This problem is eliminated by the queueing discipline used for the data packet queue. Only one packet per user is allowed on the queue at any time, and the queue is serviced on a first-come-first-served (FIFO) basis. The prevention of more than one packet per user on the queue is handled in conjunction with user flow control, discussed below.

Turnaround delay

A delay function is used by the MENEHUNE to count off the time required by half-duplex user nodes to switch from a transmit to a receive state. The actual time is determined by the equipment type—the original off-the-shelf equipment required 100 milliseconds due to its use of mechanical relays; approximately 10 milliseconds is counted off for newer equipment now in use.

Repeater scheduling

The addition of repeaters to the system introduces a number of new problems into the F_2 channel, both because of radio range overlap and the nature of the repeaters themselves. The latter are store-and-forward devices; a packet which is to be repeated is first received and stored in its entirety, then transmitted on the same frequency on which it was received (preventing reception of a new packet during this time). In order to prevent the loss of a second packet destined to the same repeater, the MENEHUNE must therefore appropriately schedule the packets in its F_2 channel queues.

For efficient scheduling (i.e., to maximize channel utilization), the MENEHUNE must know the repeater routing paths for each user node. This function could thus become quite complicated or even not achievable, depending on the degree of dynamic routing used. Because of the small percentage of traffic currently handled by repeaters in the present ALOHANET, a very simple brute force method is used: whenever a packet is sent which is forwarded by one or more repeaters, the MENEHUNE counts off sufficient time for it to be repeated once before beginning a new transmission to any node (knowledge of which packets are to be repeated is available from the user address, discussed below). This results in wasted channel capacity, but is not significant due to the capacity available in the system at present.

Packet length

Three factors having an important impact on the system are the use of variable or fixed-length packets, the way packet length or the number of data bytes is indicated, and the maximum packet length allowed. The choices made must take into account the different traffic characteristics generated by line-oriented and character-oriented user-computer interactions.

Line transmissions

Fixed-length packets were used in the initial system to simplify the design and construction of system hardware. The data packet length for both channels was chosen to allow up to 80 data bytes (640 bits), based on the user delays introduced by the 9600 bps channel data rates, the line length of the terminals in the system, and the line-oriented characteristics of the IBM 360/65 used as the central time-sharing system. An end-of-line (EOL) indicator consisting of eight zero bits was used within the packet to identify the end of actual data, where the latter was restricted to 7-bit ASCII with the eighth (parity) bit set to one. Since it was anticipated that many of the lines typed by users would be less than 40 characters, a second packet type was also defined which contained a 40-byte data field (a "Half-Packet"). This last step proved to be a mistake—the half-packet logic at each end of the link was a significant source of both hardware and software bugs.

The packet formats have since been changed to allow the use of variable-length packets with newer user nodes. An 8-bit count field is used in the packet header to indicate the number of 8-bit data bytes in the packet, with the data parity word immediately following the last data byte. In addition to eliminating the wasted channel capacity of the fixed-length packets, this also removes constraints on the data themselves necessitated by unambiguous detection of the EOL indicator within the data stream. The 80 data-byte maximum has been retained for both channels, since it still appears to be a reasonable upper bound with respect to both the multiplexing delays introduced to either channel and node buffering requirements. This should not be construed as an indication that this length is optimal, however; as file-oriented messages are introduced to the total traffic and/or user node storage continues to become cheaper, a larger maximum may be desirable for one or both channels (for a given channel data rate and user response time constraints).

Character-by-character

The increased flexibility provided by PCU's has allowed the introduction of a 'short' data packet in which a single data byte is sent in the header in place of the byte count, followed only by the header parity word. Although a use for this packet occasionally arises for interactions with line-at-a-time systems, its main use is with the character-oriented ARPANET computers now available to ALOHANET users.

The use of these character-oriented systems can have a considerable impact on the size and frequency of packets sent in the random access channel. This has an important consequence for the buffering strategy and choice of packet length used at each node: since a new transmission cannot begin until an ACK has been received for the last one, all characters typed by the user during the ACK waiting time should be sent in a single packet. Thus if communication delays tend to overlap inter-character generation times, the affected characters are accumulated at the originating node and sent (more efficiently) in a variable-length packet, without adversely affecting user-computer interaction.

A logical extension of this last strategy is to buffer all characters typed by the user at his node until one is typed which causes some action to be taken by the computer. If the appropriate set of action characters is known at the user node, this allows an optimum use of both channel capacity and system buffering without degrading the user-computer interaction. A scheme which allows this to be done in conjunction with echoing control is given by Davidson,[12] and is currently being introduced into selected ARPANET hosts. Its implementation cost in ALOHANET PCU user nodes appears reasonable, and is anticipated for use as its support by host computers becomes widespread.

Addressing

User nodes

User addressing is determined by the radio channel configuration and associated multiplexing technique. Ignoring repeaters for the moment, the two-frequency configuration used in the ALOHANET allows only a single destination in the random access channel (the MENEHUNE), and a single source in the broadcast channel (the MENEHUNE). Thus only the sender's address is required in the random access channel and only the destination address in the broadcast channel, which in both cases is the user address. Concentration of more than one user at a radio node is handled by permanently allocating a block of user addresses to the node, allowing user node multiplexing without introducing another level of addressing complexity to the system. The required address space is determined by the total number of users expected to be supported by the random access channel, and is 2^8 (eight header bits) for the present 9600 bps ALOHANET channel.

Repeaters

The use of repeaters in the system introduces some significant new factors to be considered in choosing an address scheme. Because of radio range overlap and the store-and-forward nature of the repeaters, problems can arise involving conflicts generated by two or more repeaters repeating simultaneously to the same destination, infinite repeating of the same packet (looping), and weak-signal operation due to multiple (but time-sequential) paths. In addition, the addressing scheme directly affects the MENEHUNE's ability to schedule transmissions in order to maximize broadcast channel utilization, as discussed in a preceding section. The ability to eliminate or minimize these problems depends on the degree of mobility desired for user nodes and/or the repeaters themselves.

Because of the small percentage of user nodes which currently require repeaters in the ALOHANET, a simple scheme is in use based on the hardwired properties of the original repeaters built for the system. A block of user addresses is defined for each repeater, the latter repeating only those addresses in its block. The block assigned to a repeater two hops from the MENEHUNE is a subset of the block assigned to its first hop repeater. User nodes are constrained to operate within the geographic range of their 'assigned' repeater by this scheme, but the node's user address is easily changeable if a relocation becomes necessary. Since only one path choice exists between each user node and the MENEHUNE at present, the optimum path is selected by default. As the number of repeaters in use increases and existing units are replaced by programmable devices, a more flexible repeater addressing scheme is expected to be implemented.

Resource addressing

This refers to the user's choices regarding which system resource he may communicate with. The system allows users to request a connection to the campus IBM 370/158, the ARPANET, or another ALOHANET user node. This is accomplished by sending special sequences of ASCII characters in the data portion of packets to the MENEHUNE, which may either be typed by a terminal user or automatically generated. If the requested destination is available, its identification is stored in a *Connection Table* entry for the requesting user in the MENEHUNE, and the user's address stored in a similar entry for the destination. All subsequent packets from the user are passed to the stored destination and conversely, until either end requests that the connection be broken.

Two exceptions exist to this connection table routing of packets. The first are commands intended for the MENEHUNE, such as the 'connect' and 'disconnect' above. The second is a capability which allows a user to send a single packet to another ALOHANET user independently of current connection table entries. The originating user simply types a special two-character ASCII sequence followed by the destination user's address (up to three ASCII digits), followed by the desired text.

Note that in the case of a connection to another ALOHANET node, the latter's address is also the resource address. If the node's resource can service more than one user at a time (such as might be the case for a specialized minicomputer or storage device), the present addressing scheme requires either that a block of addresses be allocated to the receiving node (as in the case of a concentrator for sending), or a sub-address be sent in the text portion of every packet. The block allocation suffers from rigidity in that resource addresses cannot be reused dynamically by different users, and does not appear desirable if many such addresses must be allocated in the system.

Error control

Random-access channel

Two distinct error sources exist at the MENEHUNE receiver, the usual random noise and errors due to packet conflicts. Because of the high probability of errors due to conflicts at full loading of the random access channel, a very reliable error detection mechanism is required. To achieve this it was decided to use two 16-bit cyclic polynomial parity check words in each data packet, one following the header and a second following the data. The separate header parity check forms the basis for a highly reliable packet synchronization method discussed in another part of this paper; it also allows reliable establishment of packet length and other information prior to processing the data portion of a packet. A single header bit is also used in conjunction with the parity check for sequence numbering, allowing the detection of duplicate packets by the MENEHUNE.

Broadcast channel

Error control for broadcast channel data packets (MENEHUNE to user nodes) involves some special considerations. For efficient operation, the usual positive acknowledgment scheme in which the ACK's themselves are not acknowledged depends on a high probability of the ACK's being successfully received. However, an ACK sent from user nodes must compete with data traffic in the random access channel. At full channel loading each random access packet must be retransmitted an average of 1.7 times, which means each data packet or ACK must be sent a total of 2.7 times on the average before it is successfully received.* But in order to force retransmission of the ACK's, the data packet being acknowledged must also be sent an average of 2.7 times by the MENEHUNE—even though it was received correctly the first time! The problem is compounded by the typically high ratios of computer/user traffic which exist for most interactive systems, resulting in many more ACK's than data packets in the random access channel. This problem was "resolved" for the initial implementation by simply not sending ACK's from user nodes. Because of the high received signal strengths at the nodes, a very low error rate was anticipated; considering also that user nodes consisted only of human terminal users, it was decided that a simple error detection/user notification scheme would be sufficient.

However, this is in general not adequate when more sophisticated data transfer functions take place or significant error rates exist at user nodes. An example of the first case is the loading of programs into core storage of a minicomputer node, where manually initiated error recovery usually requires restarting the loading from the beginning of the file. In the second case, error rates can become appreciable when user nodes are located in weak signal areas caused by distance, multipath interference, or line-of-sight blocking, or in strong signal areas in which strong local noise sources also exist. To allow for these situations, an option which allows user nodes to send positive acknowledgments has been implemented. The scheme works identically to that for the random access channel, but is only used selectively with newer programmable nodes when required (it can be turned on or off by a command from the user node to the MENEHUNE). Its effectiveness is based on the relatively light existing channel loading of the system and its use by only a few of the nodes.

One solution to this problem when all traffic to user nodes must be acknowledged in a loaded random access channel is to use sequence numbering with a large modulus, sending an ACK only when the maximum sequence number is received. This approach suffers from the unpredictable nature of interactive user-computer traffic, however; if the last computer output prior to new user input is missed by the node, a potential deadlock situation is created until the user decides something is wrong and takes manual action. An additional mechanism can be used to circumvent this, such as using automatic timeouts at the user node or sending dummy traffic to the node to 'flush out' missed packets. However, the sequence numbers succeed only in reducing the number of ACK's sent in the random access channel—to eliminate the unnecessary repetitions of data packets from the MENEHUNE, it is also necessary to acknowledge the ACK. That is, the ACK sent by a user node is timed out and retransmitted until an acknowledgment for it is received, just as for data packets. If another packet is waiting for transmission to the node at this time, its transmission with the next sequence number constitutes the ACK to the ACK; otherwise, a short ACK-ACK packet is sent by the MENEHUNE. This can be easily shown to result in significantly less total channel overhead, at the expense of more complication in the node implementation.

Repeaters

We have so far ignored the effects of repeaters in this discussion on both random access and broadcast channel error control. The repeaters currently in use in the ALOHANET do not generate acknowledgments in either direction, resulting in only end-to-end acknowledgments between the MENEHUNE and user nodes as above (but with longer minimum retransmission timeouts). This choice was made for initial repeater simplicity; it has been shown analytically, however, that a hop-by-hop acknowledgment scheme is in general superior to an end-to-end scheme, at least in contexts such as ARPANET[10] and the ARPA Packet Radio effort.[13] Thus we expect to convert to a hop-by-hop scheme when the existing repeaters are replaced by programmable units and/or repeater traffic error rates require it; this area remains a relatively unexplored problem domain within the present ALOHANET implementation.

Single-channel configurations

Finally, we note that the problems discussed above concerning ACK's sent by user nodes in the random access channel are effectively non-existent if a single-frequency channel configuration is used (and propagation times are less than the shortest packet transmission times). If all nodes can hear the transmission of all other nodes, it is only necessary that nodes refrain from sending for an ACK packet time following the transmission of a data packet by any node, except for the intended receiver who sends an ACK (if appropriate) during this time. Thus ACK's are sent conflict-free, allowing a simple positive acknowledgment scheme to be used for all traffic. Note that packets sent by the MENEHUNE are treated exactly the same as packets sent by user nodes with respect to ACK's, thus also eliminating any effects due to asymmetric computer-user traffic ratios.

Flow control

The initial system

In the initial system environment of a single half-duplex time-sharing system, model 33 Teletypes, and hardwired

* This assumes ACK's and data packets are the same length; although the ACK's are in fact shorter, the resulting error rate is still very high compared to a typical conflict-free channel.

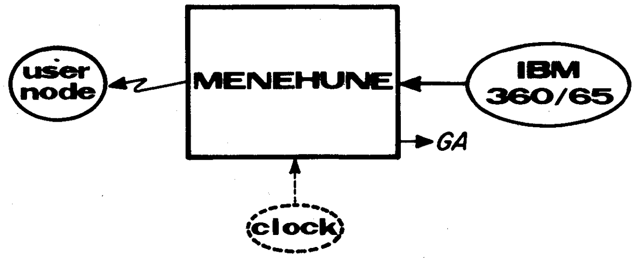


Figure 3—Broadcast Channel Flow Control (Original System)

user nodes which buffered only the line being displayed, flow control was a relatively simple matter. A user always received at least one output line from the time-sharing system (IBM's TSO running on a 360/65) for each input line, and a prompt character when it was ready for more input. The bandwidth between the MENEHUNE and 360 and the latter's I/O response times are such that one or two MENEHUNE buffers are normally sufficient to support transfers of packets received from the random access channel; in the unlikely event that no buffers are available when a packet arrives, the channel protocol guarantees its retransmission. Thus no explicit flow control was provided to prevent new packets from being sent by a user node. If the user sends one before the 360 is ready, the packet is discarded and a "WAIT" message returned to the user by the MENEHUNE (the status of each 360 connection is known in the MENEHUNE by information routinely passed from the 360).

Broadcast channel flow control was necessary, however, since each line (packet) sent to a (hardwired) user node must be completely displayed before a new line can be received. This was accomplished by the scheme shown in Figure 3, in which the control for each user node is centralized at the MENEHUNE. The latter counts off the required display time following transmission of each packet to a user, inhibiting further transmissions to that user until the time is up. To prevent 360 output from tying up MENEHUNE buffers while packets are being displayed, a handshaking flow control is used; the 360 sends only one line of output for each user, then waits for a *go-ahead* (GA) message with that user's address. The GA is sent by the MENEHUNE whenever a user's display time is up, resulting in at most one buffer required for each user (the MENEHUNE can also hold up acceptance of any packet from the 360 indefinitely until it has buffer space available). Note that this strategy also prevents any user from hogging the broadcast channel, since it allows only one packet per user in the channel queue.

Some terminal complications

The introduction of high speed CRT and hardcopy terminals to the system required an expansion of the MENEHUNE's flow control mechanism for the broadcast channel. A set of display rates was added, with the rate used at each user node stored in a permanent table in the MENEHUNE; a user can change the stored value for his node by typing a special command to the MENEHUNE at any time. The CRT terminals require an additional flow control mechanism to suspend output when the CRT screen has filled, allowing the user to signal when he is ready to proceed. Thus a screensize command was created which allows users to specify a screensize of between one and 99 lines (or an infinite screensize); this value is also stored in MENEHUNE tables for each user node. A counter is maintained for each user with a finite screensize specification and is updated for each line sent to the terminal; when the maximum is reached, the MENEHUNE suspends generation of the GA message until the user inputs a carriage return.

Satellite complications

The next complication to MENEHUNE flow control processing was caused by the connection of the ALOHANET to the ARPANET. The latter involves a 50 Kbps INTELSAT IV satellite path connecting Hawaii to California; because of its long propagation time (approximately 0.25 seconds) and ARPANET flow control protocol, a large amount of buffering is required at the receive end of the link to support continuous display at higher speed terminals—in particular, a 9600 bps terminal requires approximately a 1000-byte buffer. (Since in general CRT terminal users do not require continuous output at this rate, a smaller amount of buffering is in fact used.) This required a substantial increase in the size of the MENEHUNE buffer pool and a more complicated queueing structure to support the broadcast channel, since now more than one packet per user must in general be stored in the MENEHUNE during display at the user node. To maintain the single-packet-per-user policy for the channel queue, a separate queue was created for each user to hold additional packets. The resulting flow control scheme is shown in Figure 4, where the GA's sent to the 360 in Figure 3 are now sent to the internal ARPANET protocol module. The maximum allowed size of each user queue is determined by the user's terminal rate and the available MENEHUNE buffer pool, and in turn defines

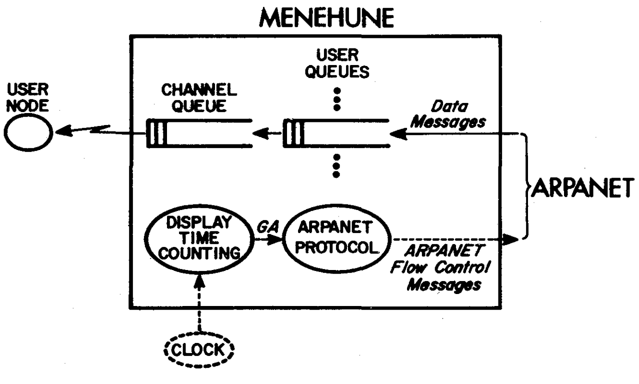


Figure 4—Broadcast Channel/ARPANET Flow Control

the parameters used in the ARPANET flow control protocol.

Multiple-line packets

A second complication resulting from the ARPANET connection concerns the extra time required by some higher speed displays for certain characters such as carriage return (CR) and/or line feed (LF). Output from the 360 in the initial system contained such characters only at the end of a line (packet), allowing the transmission time and other inter-packet delays to provide any extra time required. However, many ARPANET computers are character-oriented, at times generating many CR and LF characters within a single packet. Thus it was necessary to provide a padding function in the MENEHUNE which inserts dummy characters or otherwise adds a display time delay after each CR or LF occurrence within packets destined for a higher speed (greater than 110 bps) terminal. This necessitates the splitting of packets whenever the maximum 80-byte packet length is exceeded, and in general involves a significant amount of additional processing per packet.

Full duplex interaction

A third complication arising from many ARPANET computers is their full duplex user interaction. Unlike the 360, users do not necessarily receive output in response to each input or an indication of when the computer is waiting for more input. Since no explicit flow control is provided for input from user nodes to the MENEHUNE, users are forced to either interact in a half duplex fashion (guessing as to when the computer has finished its output) or suffer occasional losses of input data and subsequent retyping. The latter can occur frequently with the hardwired TCU's, since they contain a single buffer which is used for both keyboard input and display; if computer output arrives while the user is typing, the typed characters are overwritten in the buffer by the received packet. The newer programmable user nodes now in the system provide full duplex buffering for the terminal, allowing a packet to be received and displayed without disturbing the keyboard buffer.

However, even if user nodes are completely full duplex a flow control problem exists for packets sent to the MENEHUNE. Unlike the case for the 360, users of full duplex hosts may generate successive input packets without receiving responses from the host computer. If the ARPANET or host computer or both slow down, an excessive number of buffers can become queued in the MENEHUNE on behalf of the user. Thus, to prevent user hogging of the buffer pool a count of the number of input buffers queued for each user is now maintained; when equal to the maximum allowed, arriving packets are discarded and a discard notification returned to the user.

File traffic

The original ALOHANET design was based on a homogeneous population of terminal users generating bursty traffic into the random access channel. However, the connection of minicomputers and other terminals with memory has introduced at least two sources of non-bursty, or 'file', traffic. The first case occurs when users desire to transfer data from a paper tape or other storage media to a host computer. The second occurs when it is desired to transfer program-generated output from a minicomputer at a user node to a display device at a second user node (users can connect to other user nodes through the MENEHUNE in the same way as to the 360 or ARPANET). In either case the resulting traffic must be prevented from hogging or degrading the random access channel, and must also be constrained to the destination's acceptance rate.

The random access technique itself implicitly provides an anti-hogging mechanism, since retransmission timeouts can be used to decrease the user's average rate if conflicts occur. This does not provide for destination flow control, however, and is not necessarily an optimal solution for the random access channel. A second approach is the use of explicit flow control in the form of GA's sent by the MENEHUNE to the sending user node. This provides a solution to both problems at the expense of a small percentage of broadcast channel capacity. Since the MENEHUNE receives GA's from the user's destination, either explicity from the 360 or ARPANET module or from its display time counting for another ALOHANET node, it can simply relay them to the sending node in a short control packet. This approach also allows centralized optimization of traffic in the random access channel by the MENEHUNE, and is the subject of current investigation.

RADIO SUBSYSTEM CHOICES

The design of the ALOHANET radio communication system required the balancing of a number of performance goals against various system constraints which are peculiar to the use of radio frequencies for data communication channels. These trade-off studies resulted in the selection of our RF channels and modulation method. The determination of operating ranges and the choice of a data synchronization method resulted from the basic channel and modulation selection decisions. In this section we will describe the primary issues related to RF channel selection, modulation design, radio range determination, and data synchronization design.

RF channels and modulation

The choice of radio channels for any communication system is a complex task, requiring the trade-off of many factors such as desired bandwidth, area coverage, spectrum availability, potential interference and noise

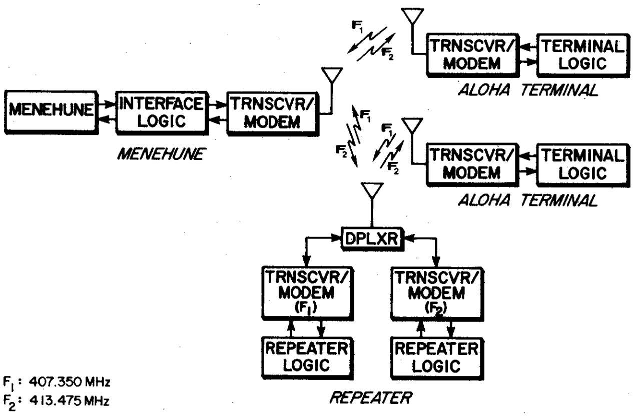


Figure 5—ALOHA System UHF Radio Communication System

sources, regulatory requirements, and equipment costs. In the case of the ALOHANET, a wide channel bandwidth was considered desirable for the random access channel since user nodes are required to send messages to the MENEHUNE at high peak data rates compared to their average data rate. Wide bandwidth was also deemed advisable for the broadcast channel due to the expected high traffic density from the MENEHUNE. The use of wide channel bandwidth tends to force the use of higher frequencies where spectrum crowding is less severe and the availability of bandwidth is greater. Crowded radio bands are undesirable not only from the standpoint of interference to other users but also because of potential interference from them. Another disadvantage of lower frequencies is the higher probability of interference from man-made noise sources, particularly in an urban area where the ALOHANET has most of its terminals.

From the above considerations it can be seen that the system's communication requirements tend to emphasize the use of higher radio frequencies. The primary constraint on moving to even higher frequencies is equipment cost and radio range. Above 500 MHz equipment costs tend to escalate rapidly. Area coverage also becomes more difficult due to more pronounced shadowing effects of the radio waves by buildings and hilly terrain. (Above 30 MHz radio propagation tends to be limited to line-of-sight paths.)

Therefore, the 400 to 500 MHz UHF band was selected as the optimum for the ALOHANET radio frequencies. Reasonably priced commercial radio equipment was found to be available in this frequency region and radio band crowding was not severe in Hawaii. Initially, assignments in the 450 to 470 MHz mobile radio band were requested but were rejected by the FCC because of our wide channel bandwidth requirements. (The mobile radio channels are specified at about 15 KHz bandwidth, whereas we were requesting 100 KHz.) We were fortunate enough to receive assignments as an experimental service in the government UHF band of 406 to 420 MHz, where spectrum space was available.

Since most radio equipments available in the UHF bands use frequency modulation (FM), this type of modulation was selected for the RF channels. A slight variation was incorporated in the hardware design to minimize the interface problems between the radios and the data modems. This variation was the use of a subcarrier tone to carry the actual data modulation. This tone is phase-shift-keyed by the data and the resultant signal is used to modulate the FM transmitter. This modulated tone is recovered from the FM receiver and fed to the demodulator of the modem. This modulation system is referred to as FM/DPSK to indicate frequency modulation by a differentially phase-shift-keyed subcarrier. (Differential phase-shift-keying is used to resolve the problem of received phase ambiguity.) The resultant configuration is shown in Figure 5.

Radio range

The maximum operating distance between any terminal of the ALOHANET and the MENEHUNE (or a repeater) is specified as the system's radio range. This distance is primarily a function of a transmitter's radiated power, the receiver's sensitivity, and the attenuation of radio signal power for the given distance. Local noise conditions at the receiver location can also affect this distance, but for system planning purposes, range is usually calculated on the basis of some given propagation model. For line-of-sight paths, which exist at VHF, UHF, and higher frequencies, two different models are used depending upon local topographical conditions. In an urban area these paths are partially obstructed and suffer from multipath effects. A power loss proportional to $1/R^4$ is usually assumed for these conditions.[14] Where paths are unobstructed and well clear of the local terrain, a spreading loss proportional to $1/R^2$ can be assumed. Receiver threshold sensitivity in the ALOHANET is defined as that receiver input power level which causes an average bit error rate of 10^{-5}. This bit error rate should provide a packet throughput reliability better than 99 percent for full-length ALOHA packets.

Assuming a transmitter equivalent radiated power of 10 watts, a simple whip antenna at a user terminal, an elevated antenna at the MENEHUNE or repeater and a 3 microvolt receiver sensitivity, the radio range works out to about 17 miles in the urban area for the ALOHANET frequencies. Between repeaters and the MENEHUNE terminal, which have well-elevated antennae and good path clearances, the assumed $1/R^2$ model gives a maximum range of 290 miles. The use of high-gain omnidirectional antenna arrays at repeater sites extends these ranges. Tests conducted on a 100 mile path between two ALOHANET repeaters confirmed the $1/R^2$ spreading-loss assumption and indicated a fade margin of 30 db existed (due to the 10 db gain antennae used for the test).

Data synchronization

Because of the burst nature of radio transmission of ALOHANET packets, special synchronization techniques must be employed in the modem and data terminal equipment. Since the phase-shift-keying used in the ALOHANET modem design is a bit-synchronous technique, bit synchronization must first be performed in the demodulator before packet synchronization can be attempted. Bit-sync is performed by a phase-locking circuit, and a lock-indication signal is passed to the data equipment when bit-sync has been attained. The bit-sync detection circuit is so designed to provide a very low false detection probability (less than 10^{-6}) and a high probability of packet detection. The narrow bandwidth of the phase-lock circuit presently designed into the ALOHANET modem requires a bit-sync preamble of 90 bits to ensure reliable bit-sync. Studies have indicated that this preamble can be reduced to about 10 bits by use of a redesigned wide-band phase-lock circuit. In fact, we are presently contemplating doing away with the bit-sync preamble entirely, further reducing packet overhead. The unique characteristics of the ALOHA modem design make such an approach feasible.

Packet synchronization is accomplished in the ALOHANET data terminal buffer by means of the 16-bit parity word contained in the packet header. When the parity check routine accepts the header, the packet is assumed to be synchronized. Since the parity check routine is initiated by the first bit of the header, packets can be missed due to detection of an early error bit before the header. This miss probability is presently controlled by the modem at about 10^{-3} or less, providing a packet detection probability of 99.9 percent or better. The false detection probability of this circuit is $\sim 1.5 \times 10^{-5}$, which is independent of that of the modem. Thus, the overall probability of false detection is less than 1.5×10^{-11}. Therefore, less than one out of a thousand packets will be lost due to packet sync errors and packet sync false alarms occur with extreme rarity.

USER INTERFACE CHOICES

The development of the ALOHANET user interface has been an evolutionary process, as is typical of most research developments. Since there were expected to be many user nodes (as compared to the single MENEHUNE node), the primary design goals were initially set as simplicity of design and low cost. This led to the design of a hardwired control unit with limited data storage capability coupled to a modem and radio transceiver. This initial design was termed a Terminal Control Unit (TCU). As experience developed with operation of the net, other functions became evident as being desirable in a TCU. At about this time the first microprocessor chips and low-cost semiconductor memory chips were becoming available in the marketplace. It was decided that a new TCU design should be initiated using these new devices since much greater flexibility and additional functions could be readily incorporated in a unit having a capability of being programmed. It was also noted that the cost of these new devices was such that a unit could be built for the same cost or less than that of the original design. Thus, the Programmable Control Unit (PCU) was developed, and there are now several operating units in the system. We will now discuss some of the issues involved in designing a terminal control unit for use on the ALOHANET. These issues lie in the general areas of interface considerations, and the technology of microprocessors.

The original TCU

The ALOHANET was originally envisioned as a terminal network, with the TCU's interfacing human users to a half duplex, line-oriented time-sharing system. At the time of the first TCU design effort memory was relatively expensive, so in order to minimize cost a single buffer was chosen for use with both the terminal keyboard and display. (As noted earlier in this paper, when full duplex computer interactions were available in the system the single buffer was found to be quite a disadvantage.) The buffer was designed for a full line length of 80 characters, which allowed handling of both the 40 and 80 character fixed-length packets defined for the system.

Additional basic functions performed by the TCU's were generation of a cyclic-parity-check code vector and decoding of received parity code words for error-detection purposes, and generation of packet retransmissions using a simple random interval generator. If an acknowledgment was not received from the computer after the prescribed number of retransmissions, a flashing light was used as an indicator to the human user. Since the TCU's did not send acknowledgments to the MENEHUNE, a steady warning light was displayed to the human user when an error was detected in a received packet. Thus it can be seen that considerable simplification was incorporated into the initial design of the TCU, making use of the fact that it was interfacing a human user into the network.

Other functions hardwired into the TCU were the obvious requirements of checking for and generating its address, packet sequence numbering, checking to see if a received packet is an ACK packet or a data packet, and generating and checking for half- or full-packet conditions. (The control bits for these functions all reside in the header portion of the packet.)

The final consideration was the choice of standard interface signals between the TCU and the user's equipment. This was a relatively simple choice, since most equipment is designed to meet the EIA standard RS 232C interface specification. Therefore, the TCU was designed to meet this standard, which allows direct connection of most terminals in use today.

Minicomputer nodes

As the ALOHANET developed, some minicomputers were interfaced into the network as concentrators for a number of terminals. Many of the logical functions performed in a TCU were now incorporated into the mini's software, with error detection and parity word generation performed in a special hardware interface unit imposed between the minicomputer and an ALOHA modem. (This unit was very much like the encoder/decoder unit used at the MENEHUNE to interface that minicomputer to the channel.) Parallel-to-serial and serial-to-parallel conversion was also performed in this interface unit.

However, a minicomputer is an expensive device to use for these simple functions, and it requires considerable amounts of power and space. If it already exists for the purpose of performing various user-oriented tasks, then it is cost-effective to incorporate the software interface and a minimal amount of hardware for use on the ALOHANET.

The advent of the microprocessor chip changed all this. The relatively low-cost processing power demonstrated by these units made it apparent that many system options we had previously considered and discarded because of hardware complexity and cost limitations in the TCU, were now viable in a PCU. Some of these options—file transfer, remote user ACKs, single frequency operation, character-by-character transmission—were discussed in previous sections. This trend toward programmable and more powerful TCU's has thus led to the development of the ALOHA PCU, using a microprocessor to handle the TCU buffering and control functions, in addition to more complex and sophisticated functions.

Microprocessor technology

The development from the hardwired TCU concept to the fully-programmable PCU has closely followed the rapidly changing technology of microprocessors. The availability of lower-cost semiconductor memory has allowed the evolution from half-duplex to full-duplex operation in the PCU, with the beneficial side-effect of decreased logical complexity due to separation of the input and output functions. However, the first PCU developed had a hardware complexity level comparable to the TCU due to the relatively primitive structure of early microprocessor designs. This first PCU, designed with the Intel 8008 CPU, required a considerable amount of circuitry for buffering and multiplexing functions needed with this early microprocessor chip. Because of the slow speed of the chip, bit-by-bit processing was not possible and additional buffering was also necessary. But, much greater flexibility was introduced into the scope of functions which could be performed, due to its programmability.

Later microprocessor designs, such as the Intel 8080 and National IMP-16, have introduced much greater sophistication into the processor chips accompanied by significant processing speed improvements. A newer PCU design, incorporating an Intel 8080 chip, has demonstrated a considerable reduction in hardware complexity accompanied by an even greater degree of processing flexibility. For example, parity generation and checking are done in software with this prototype design.

Buffering has progressed from the simple shift-register storage devices of the TCU to the use of semiconductor RAM devices used in the microprocessor's random-access memory. All of the micro-instructions for the Intel 8080 microprocessor PCU design reside on four PROM chips, providing 1024 bytes of microcode. The random-access memory consists of 2048 bytes of RAM.

Recent product introductions such as Intel's 3000 series bi-polar chips promise even greater reductions in chip counts and increases in processing power and speed. With machines such as these, bit-by-bit processing can be readily incorporated into software, thus further eliminating the need for external interfacing hardware and simultaneously providing greater flexibility in the implementation of additional functions. A more detailed discussion of communications microprocessors is given in a companion paper in these proceedings.[9]

Size and power

In the earlier versions of the TCU smaller size and power drain of the unit were not considered major design objectives. The first units were designed for ease of access and hardware modifications to these TCU's were made on a fairly casual basis. As more and more of the ALOHANET came into use, however, small size, portability and lower power drain became desirable.

Of particular interest is the possibility of designing low power battery operated portable PCU's for mobile units in the ALOHANET. Since the transmitter power need only be on for a short burst corresponding to the period of the data burst, the average power of the transmitter can be a small percentage of the peak power. Since low power and small size were not original design objectives, it appears that the construction of low power portable PCU's will involve redesign of several subsections of the PCU and some new design efforts. Of particular importance is selection of a microprocessor unit which provides a minimum power-drain computer architecture consistent with functional requirements. The modem should be redesigned to use MOS devices to minimize power drain, and the transceiver designed for minimum complexity.

CONCLUSIONS

As the system has been modified during the past several years it has become apparent that packet broadcasting architecture is remarkably flexible in its tolerance of hardware, system and protocol modifications. This flexibility follows from the packet verification algorithms which lie at the basis of packet broadcasting. The only packets accepted by a remote unit or by the MENEHUNE are packets which meet all the tests expected by the potential acceptor; and the only system resource consumed by an unaccepted packet is the capacity of the channel during the short burst of the packet duration. Thus it is perfectly feasible in a packet broadcasting network to introduce a new form of packet (new in format, new in packet length, or even new in modulation technique) without disturbing any unit operating with the existing scheme. Only the units designed to look for the new packets will accept these packets and all other units will simply discard them.

We plan to employ this property of packet switched channels to switch the polynomial used for error control in the present packet format. The new polynomial is available in a single IC chip and will allow the possibility of error correction as well as error detection in some cases. As remote units with new packet formats are put into operation we can continue to operate the existing remote units without modification as long as we have a single unit capable of accepting the new packet format at the MENEHUNE. As a side benefit of the introduction of this modification we also note that we have effectively doubled the number of user addresses in the system. An address in use with the old packet format may be reused with the new, since each is effectively invisible to the other.

Another result of our ALOHANET experience, current technology, and recent theoretical work on ALOHA channels, is that a *single-channel* network configuration appears preferable to the two channels used in our present system. The major reason why this is so has to do with the broadcast property of the single-channel system, in which all nodes can (for a given geographic range) hear the transmission of all other nodes in the net.

A number of desirable properties result from this broadcast feature. First, each node can determine if the channel is free before transmitting, greatly reducing the number of packet conflicts—Kleinrock and Tobagi[15] have shown analytically that this can increase the throughput of a random access channel by a factor of three to five for reasonable user delays, depending on the propagation times between nodes. Second, the problem of sending acknowledgments from user nodes is resolved in a simple manner. Third, system bandwidth can be optimally allocated to both directions of traffic by simple time-sharing of the channel. Fourth, single channel repeaters require only half the radio hardware of two-channel repeaters, and, in fact, the radio transceivers at all nodes need be only half duplex. Finally, a single-channel system constitutes a fully-connected network allowing direct communication between all nodes. A star configuration can still be imposed by protocol to direct all user traffic through a central node, but is no longer required.

It is important to note that many of the above properties are made feasible by the availability of PCU's at a reasonable cost through microcomputer technology. This raises a related issue: the desirability of distributing presently centralized protocol functions such as flow control among the user nodes. Since we have just begun to gain experience with PCU's in a packet broadcast network, we must leave this as an open question.

REFERENCES

1. Roberts, L. G. and B. Wessler, "The ARPA Computer Network," in *Computer Communication Networks* edited by Abramson and Kuo, Prentice-Hall, 1973.
2. Kahn, R. E., "The Organization of Computer Resources into a Packet Radio Network," in this issue, *AFIPS Conference Proceedings,* Volume 44.
3. Abramson, N., "Packet Switching with Satellites," *AFIPS Conference Proceedings,* Volume 42, 1973, pp. 695-702.
4. Binder, R., W. S. Lai and M. Wilson, *The ALOHANET MENEHUNE—Version II,* ALOHA SYSTEM Technical Report B74-6, University of Hawaii, September 1974.
5. Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther and D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," *AFIPS Conference Proceedings,* Volume 36, 1970, pp. 543-567.
6. Ornstein, S. M., F. E. Heart, W. R. Crowther, H. K. Rising, S. B. Russell and A. Michel, "The Terminal IMP for the ARPA Computer Network," *AFIPS Conference Proceedings,* Volume 40, 1972, pp. 243-254.
7. Jackson, P. E. and C. D. Stubbs, "A Study of Multi-Access Computer Communications," *AFIPS Conference Proceedings,* Volume 34, 1969, pp. 491-504.
8. Abramson, N., "The ALOHA System," in *Computer-Communication Networks* edited by Abramson and Kuo, Prentice-Hall, 1973.
9. Fralick, S. C., D. H. Brandin, F. F. Kuo and C. Harrison, "Digital Terminals for Packet Broadcasting," in this issue, *AFIPS Conference Proceedings,* Volume 44.
10. Metcalfe, R. M., *Packet Communication,* Project MAC Technical Report, MAC TR-114, MIT, July 1973.
11. Lam, S. S., *Packet Switching in a Multi-Access Broadcast Channel with Application to Satellite Communication in a Computer Network,* UCLA-ENG-7429 Report, University of California at Los Angeles, Computer Science Department, April 1974.
12. Davidson, J., *An Echoing Strategy for Satellite Links,* NIC Document 10599, (RFC 357), Stanford Research Institute, June 1972.
13. Frank, H., R. M. Van Slyke and I. Gitman, "Packet Radio Network Design—System Considerations," in this issue, *AFIPS Conference Proceedings,* Volume 44.
14. Okumura, Y., E. Ohmori, T. Kawano and K. Fukuda, "Field Strength and its Variability in UHF and VHF Land-Mobile Radio Service," *Review of the Electrical Communication Laboratory,* Vol. 16, No. 9-10, September-October 1968.
15. Kleinrock, L. and F. Tobagi, "Packet Switching in Radio Channels: Carrier Sense Multiple-Access Modes and their Throughput Delay Characteristics," to be published in *IEEE Transactions on Communications.*

Packet radio system—Network considerations*

by HOWARD FRANK, ISRAEL GITMAN and RICHARD VAN SLYKE

Network Analysis Corporation
Glen Cove, New York

INTRODUCTION AND NETWORK OVERVIEW

The Packet Radio System[1] considered as a network is characterized by devices, terminals, repeaters and stations, linked together by broadcast radio channels.

The main features which distinguish the Packet Radio System from a point-to-point packet switching system (such as the ARPANET) are: (i) devices in the system transmit packets by using a random access scheme, and (ii) devices broadcast so that packets can be transmitted to several devices simultaneously, and/or several packets can be simultaneously received by a receiver because of independent transmissions of several devices. These features have a major impact on practically every aspect of network considerations.

There are three basic functional components of the Packet Radio System: the Packet Radio Terminal, the Packet Radio Station, and the Packet Radio Repeater. (See Figure 1) Packet Radio Terminals may include diverse devices such as hand held mobile terminals, TTY-like and CRT terminals, unattended sensors, computers (micro, mini, and maxi), display printers, and position location devices.

In some applications the Packet Radio Station will be the interface component between the broadcast system and a point-to-point network. As such it will have broadcast channels into the Packet Radio System and link channels into the point-to-point network. In addition, it will perform accounting, buffering, directory, and routing functions for the overall system.

The basic function of the Packet Radio Repeater is to provide a network for connection of terminals to one or more stations, thereby increasing the size of the area that can be served by a station and providing paths to alternate stations to insure reliable communications. A more detailed discussion of the network hardware functions can be found in the next section.

The devices (repeaters, stations, and terminals) of the Packet Radio System communicate in a broadcast mode using random access techniques.[2] The configuration of the broadcast channel is discussed later—where it is shown that unless the traffic rates from station to terminals as compared to the traffic rates from terminals to station are known quite accurately, it is better to share the channel between the two directions rather than divide the channel into two channels: one for traffic to the station, and another for traffic away from the station. Means for alleviating the large concentration of traffic near the station offers large potential benefits. Two approaches to achieve these benefits are: (1) adding directional antennas on the repeaters next to the station for the repeater station link to minimize interference, and (2) using multiple directional antennas at the station to communicate with adjacent repeaters. These approaches are also evaluated in the fourth section of this paper.

The number of stations within the network is determined by the amount of traffic to be handled. Thus, to first approximation, we can think of partitioning the area to be covered into regions of equal traffic and allocate one station for each region. In regions of low traffic density, the station may not be in "line of sight" of all the terminals in the region; hence repeaters are used to relay the traffic to the station. Thus, repeaters correspond to a geographical partition of the area into sections small enough so that each terminal can communicate with a repeater and be relayed by it to a station. Network topology, the location of devices (terminals, repeaters, and stations) and the interconnection between them on the broadcast channel, is discussed in more detail later.

The principal goals of communication in the Packet Radio Network are to send packets through the network (1) efficiently and (2) correctly. The first objective is intimately related to the routing procedures used and the second to the acknowledgment mechanisms used. These two topics are covered in a later section. Of particular interest is a hierarchical directed routing algorithm.

The broadcast transmission equipment, common to repeaters stations and terminals in the experimental system, has the capability of sending at two data rates and with adaptive (variable) transmission power. In the last part of this paper, the studies leading to the incorporation of these features in the units are discussed.

In areas of high traffic, such as urban areas, repeaters may not be needed: in fact, the problem may be that a station can communicate with more terminals than it can handle. Broadcast of data in urban areas is also complicated by multipath interference.[3] The rapidly expanding Cable Television (CATV) Systems within urban areas offer an attractive alternative to over-the-air broadcasting, except for mobile users who must use broadcast techniques. The same

* This work was supported by the Advanced Research Projects Agency of the Department of Defense under Contract Number DAHC 15-73-C0135.

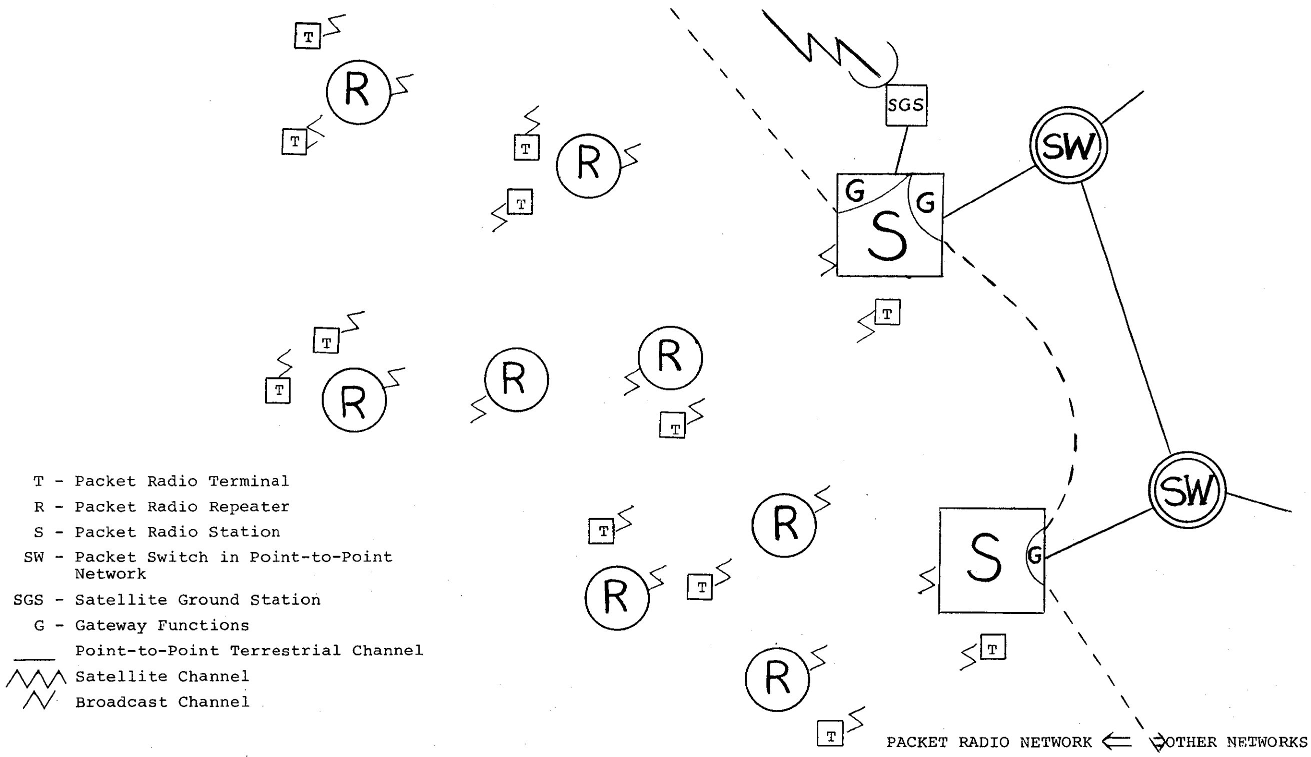

T - Packet Radio Terminal
R - Packet Radio Repeater
S - Packet Radio Station
SW - Packet Switch in Point-to-Point Network
SGS - Satellite Ground Station
G - Gateway Functions
Point-to-Point Terrestrial Channel
Satellite Channel
Broadcast Channel
T
R
S
G
SW
SGS
PACKET RADIO NETWORK ⇐ ⇒ OTHER NETWORKS

Figure 1—Packet radio network

general Packet Radio concept can be applied to Packet Communication on CATV systems. This approach is discussed in Reference 4.

The techniques used to investigate the problems discussed in this paper were analytic modeling and simulation, applied to specific system configurations. The design decisions and performance characteristics projected using these techniques will be evaluated by a sequence of studies being planned for the experimental packet radio system.

NETWORK DEVICES

In this section we discuss the devices' functional capabilities which are necessary for *communication* in the Packet Radio network.[5–8] Functional requirements of elements not directly related to communication are not discussed.

Terminals: There are two categories of terminals; (i) those which usually await a response to a message they transmit (e.g., manually held radio terminals, small computers), and (ii) those which do not require such responses or acknowledgments (e.g., unattended sensors, position indicators). Some terminals in the former category will usually send and/or receive several packets in one message.

Necessary or desirable communication capabilities of a terminal:

(1) Ability to identify packets addressed to it.
(2) Calculation of packet checksum.
(3) Capabilities related to packet routing such as; retransmitting packets when acknowledgments are not received, recording and using a specific ID of a repeater and/or station to be used for other packets of the same message, counting the number of retransmissions.
(4) Capabilities related to the response to previously determined types of error.
(5) For unattended terminals, capabilities by which a centralized control or a station will be able to identify whether the terminal is operative or dead.

Repeaters: Requirements for repeaters include:

(1) Calculating packet checksum.
(2) Packet storage and retransmission.
(3) Capabilities by which a station can determine whether a particular repeater (or any repeater in a particular area) is operative or dead.
(4) Capabilities (1), (3), and (4) of terminals.
(5) Capabilities, dependent on the routing strategy, for calculating the most efficient next repeater on a transmission path to a station or to a terminal.

Stations: Among the stations' requirements are:

(1) A dynamic directory of active terminals and repeaters in its region.
(2) Gateway functions necessary to transfer packets between the Packet Radio System and another network.

(3) Storage buffers for packets received from terminals and for packets to be transmitted to terminals.
(4) Storage for character position information for active terminals which do not have this capability.
(5) Accounting capabilities.
(6) Capabilities related to routing flow control, and network management.

NETWORK TOPOLOGY

General considerations

Many factors affect the location or repeaters and stations. Simple consideration of repeaters as area covers and stations as traffic covers neglects interactions between the two types of devices.

Factors affecting the location of repeaters and stations in addition to range and traffic are:

(i) *Logistics*: Some locations for repeaters may be preferable to others because of greater accessibility or more readily available power, eliminating the need for batteries (e.g., on telephone poles or near power lines).

(ii) *Reliability and redundancy*: For many reasons, redundant repeaters and stations will be required. Since repeaters in remote areas will operate on batteries, it will be necessary to have sufficient redundancy so they need not be replaced immediately. Stations and repeaters will have intermittent and catastrophic failures for which backup is required. Extra repeaters are needed when line of sight to the primary repeater is locally blocked.

The devices (repeaters, stations, and terminals) of the Packet Radio System communicate in a broadcast mode using random access techniques which are suitable for terminals that: (i) are mobile, so that a broadcasting mode is necessary, (ii) are located in remote or hostile locations where hardwire connections are infeasible; (iii) have a high ratio of peak bandwidth to average bandwidth requirements (because random access methods allow the dynamic allocation of channel capacity without centralized control); or (iv) require little communication bandwidth so that hardwire connections are uneconomical.

In order to realize these potential benefits of random access techniques, effective traffic control procedures are required. The discipline chosen and its efficiency will probably be the single most important factor affecting system performance. This will prevent messages from circulating endlessly among the same group of repeaters, and from propagating to a geometrically increasing number of repeaters in a large network.

Device location

To provide line-of-sight coverage of an area where mobile terminals or fixed terminals are transmitting by radio from unspecified locations we must locate *repeaters* so that any such terminal will be in line-of-sight of repeaters and that there be reliable connections between every pair of terminals (and repeaters). To make this possible there must be several repeaters visible to each terminal. We would like to minimize the installation cost and maintenance cost of the repeaters subject to a constraint on the reliability of service.

In general, determining if line-of-sight micro-wave transmission between two points is possible, involves taking into account many factors including wave-length (Fresnel zones), weather conditions (effective earth radius), antenna design, height, topography, etc. Nevertheless there are methods for making such calculations;[9] we describe methods for using the results of the determinations to choose good locations for the repeaters.

The principal and immediate interest is in an appropriate mathematical model of the situation and some indications on how to solve the problem. The first problem is the proper choice of reliability measure or grade of service. We assume that the radio network is for local distribution-collection of terminal traffic with rates small compared to the channel capacity so that throughput capacity is not a constraint. That is, if any path through the network exists for a given pair of terminals we assume there is sufficient capacity for traffic between them. Possible measure of network reliability that have proved useful in the analysis of communication networks are given in Reference 10. However, for network *synthesis* as distinguished from *analysis* these approaches appear too difficult both from computational and data collection point of view. This suggests the "deterministic" requirement that there exist k node disjoint paths between every terminal pair. This guarantees that at least k repeaters or line of sight links must fail before any terminal pair is disconnected.

One might demand only that there be k node disjoint paths between every pair of terminals instead of between each terminal-repeater pair, but we are assuming that communication always takes place through a "station" which could be any of the repeaters. The analysis of the terminal to terminal model is similar in any case.

The constraint can be broken into two parts:

1. the repeaters must be located so that at least k of them are in line of sight with each terminal, and
2. between each pair of *repeaters* there must be k node disjoint paths.

Because the repeaters will have substantially greater range than terminals, the first aspect of the constraint will ordinarily be dominating. This problem is closely related to the classical set covering problem. Extensive research has been and is being done on this problem but there is good evidence empirical[11] and theoretical[12] that the problem is intrinsically difficult.

Given the limited success to be expected from exact algorithms in solving large scale problems, we have been led to consider heuristic methods to find good solutions to the covering of terminals by repeaters problem, which is typically large scale. It is intuitively appealing to consider a *terminal* as particularly *critical* if it is adjacent to few repeaters. (In

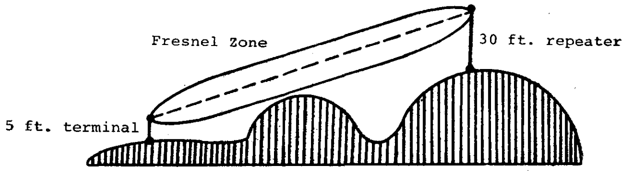


Figure 2

the extreme cases, if a terminal has fewer than k adjacent potential repeater locations, the problem is infeasible and if it has exactly k adjacent repeaters, all of them must be chosen for any feasible solution.) Similarly, a *repeater* is *desirable* if it is adjacent to a large number of terminals, especially if the terminals are highly critical. The heuristic algorithms systemize these intuitive notions in the search of a "good" solution.

The size of test problems solved varies from problems with as few as 5 repeaters and 5 terminals to problems with as many as 400 potential repeater locations and 400 terminals. Roughly speaking, the computation time is directly proportional to the size of the incidence matrix and the cover multiple required. The computer used is a PDP-10 (time sharing). The larger problems (400 repeaters, 400 terminals, 2-cover) were solved in 70 sec. or less. The time, as may be expected, is dependent on the density of the repeater-terminal incidence matrix. Thus, the maximum time recorded arose from terminal—repeater configurations where each repeater covers many repeaters. The running time is of the order of $| T | \times | R |^2$ where $| T |$ and $| R |$ are the number of terminals and repeater locations respectively.

Another test problem used to compare various techniques to solve the k-covering problem was generated by using real data obtained from a topographical map for the region of Palo Alto. This part of the U.S. was selected because it contained many interesting topographical attributes: a flat terrain (salt flats, the region surrounding the Bayshore Freeway), an urban center (Palo Alto and neighboring communities) on slightly sloping terrain and finally a hilly region (with valleys, small plateaus, etc.). Moreover, a reduced scale experiment of a packet radio network is planned by ARPA for late 1975, in the Palo Alto area.

LOS Computation: To determine if a terminal at location j can be seen from a repeater at location k, we proceeded as follows. It was assumed, that if no particular high construction (building, water tower, etc.) was available to install the repeater's antenna, it would be installed at 30 feet above the ground level (making use of a tree, telephone pole, etc.). The terminals were assumed to be 5 feet above ground level. The points were said to be in LOS if the first *Fresnel Zone* associated with transmission between these two points was free of any obstacle. To compute the Fresnel Zones, we assumed that transmission would occur at 1500 MHz corresponding to a wave length $\lambda = .2$m (7.87 in.).

The problem was solved by the heuristic algorithm and by OPHELIE—a mixed-integer programming system for the CDC-6600 computer. (A rapid analysis of the terminal-repeater adjacency matrix shows that none of the optimal solutions would have been generated if one had used the more simplistic approach of selecting the repeater with highest adjacency degree. Such a selection yields quite different answers requiring a larger number of repeaters.)

The optimal solution requires the installation of 14 repeaters (different runs with the heuristic showed that there were in fact a number of optimal solutions with 14 repeaters). The total running time for OPHELIE was approximately 12 CPU sec. excluding set up time. The heuristic required 3 sec. to produce a solution. The relative success of the OPHELIE code must, at least in part, be attributed to the fact that the linear programming solution (which is used to initiate the branch-and-bound part of the code) is actually the optimal solution.

Network reconfiguration

From the general topological considerations, it is apparent that the routing and flow control algorithms will be the main factor which will determine the efficiency of the Packet Radio System. However, there are two contradictory requirements; reliability considerations imply that every repeater should be able to transmit to several repeaters; on the other hand, efficiency considerations suggest that *one* repeater should receive and relay the packet, preferably the repeater along the shortest path to the destination. A sensible solution is to assign to the set of repeaters a structure which will transform the broadcast network to a point-to-point network for routing purposes. The problem is that the connectivity of devices is changing, and therefore, it is necessary to develop algorithms for dynamically changing the network structure (reconfiguration) under certain conditions. Examples of a changing topology are when the network is mobile (e.g., a Packet Radio System for a fleet of ships), when a repeater's range charges (e.g., from battery drainage) or when repeaters fail. A different type of network change occurs when terminals enter or leave the areas served by a given repeater or station.

In Reference 13, we propose algorithms for dynamically changing the network configuration of repeaters. It is assumed that every repeater and station has a fixed ID, and that there is a simple routing algorithm which may be inefficient, however, which is independent of any network structure. The process contains three steps;

STEP I—Mapping the network connectivity. This is obtained by a process in which stations transmit packets to repeaters, requesting each to respond with a trace packet into which every repeater along the path adds its fixed ID.

STEP II—Determining network structure. The connectivity information obtained above is used to obtain a network structure which has several properties: for example, it enables every packet to be routed along the shortest path (minimum number of hops); it determines the repeaters which are not needed for relaying packets and which should be temporarily disabled.

STEP III—In this step, the stations transmit the structure information to repeaters and tests each path in both directions.

CHANNEL CONFIGURATION

Split versus common channel

Apart from the suitability for mobile terminals, random access schemes offer an attractive alternative to fixed assignment of channel capacity (FDM, TDMA) for applications characterized by traffic of a bursty nature. (That is, when the traffic requirements of users can be characterized as having a high peak to average data rate.) This is because at any given time, the capacity assigned to non-active users is not utilized; whereas the active users experience relatively long delays due to the low data rate available to each.

We pursue this same argument one step further and investigate for the packet radio system whether we should have two channels, one for transmission from terminals to stations and the second in the reverse direction; or alternatively whether we should dynamically share the total capacity (common channel). This problem was investigated for a single hop network in which n stations communicate with an infinite number of terminals using the slotted ALOHA random access scheme.[14] In the model it is assumed that all stations and terminals are within an effective transmission range of each other, that the processes of packet originations and packet originations plus retransmissions are Poisson, and that there is a ratio α of the rate of packets which originate from stations to the rate which originate from terminals.

Figure 3 shows the comparison of the maximum effective utilization of the two configurations as a function of α with the number of stations, n, as a parameter. The subscripts s and c denote the split (into equal parts) and common configurations, respectively. The results show that for any given channel split, there is an interval for α, within which the split channel configuration is superior to the common channel configuration. Outside this interval, the common channel configuration is superior. The above interval decreases when the number of stations increases and reduces to a single point when n tends to infinity.

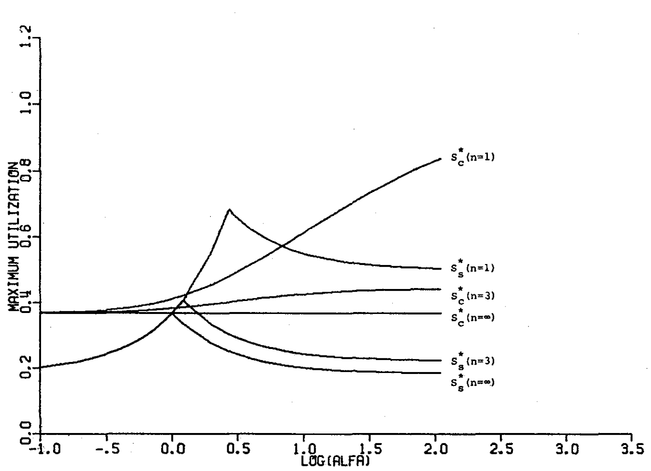


Figure 3—Maximum utilization of split and common channel configurations

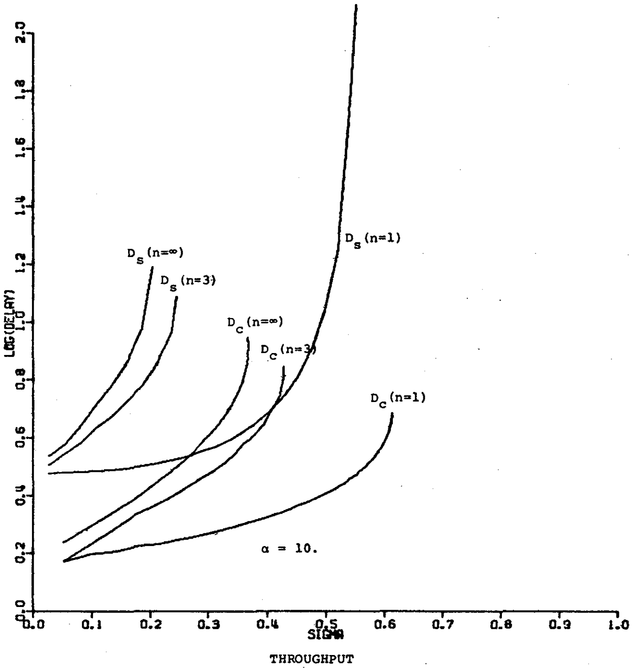


Figure 4—Delay vs. throughput, $\alpha = 10$

The conclusion from these results is that if the ratio α is not known or if it varies, it is preferable to share dynamically the total capacity. Figure 4 shows an example of the average delay of a packet in the system (weighted average of packets in the two directions) as a function of the total throughput, for the case $\alpha = 10$. The difference in the packet transmission time (slot) due to the difference in the data rates of the two configurations has been taken into account. In this case, the common channel performance is uniformly better than the performance of the split channel. Moreover, it can be shown that this superiority holds in almost all cases.[14]

Directional antennas and multiple transmitters

Another problem related to channel configuration is the possible use of directional antennas by repeaters and/or stations and the advantage (if any) of using multiple directional transmitters. This problem was investigated for a 2-hop and single station packet-radio network.[15] The investigation was done assuming separate channels from station to terminals and from terminals to station, and for the slotted ALOHA random access scheme.

Transmission from terminals to station

Consider a 2-Hop system with m repeaters and a single station as shown in Figure 5. The traffic originates from terminals and is destined to the station. A terminal transmits

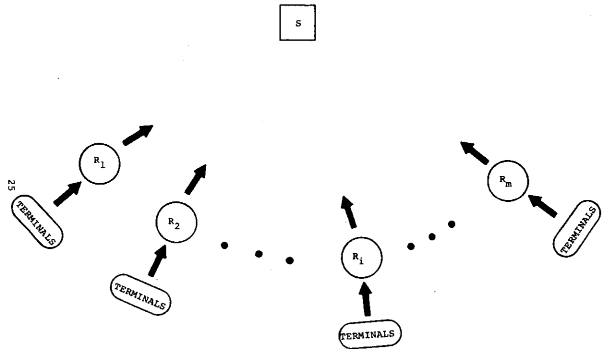


Figure 5—Transmission from terminals to station

its packets to a repeater (hop 1), which in turn transmits the packets to the station (hop 2). The transmission protocol is as follows: when a packet becomes ready for transmission, it is transmitted into the next slot; the device then times out waiting for an ack, and if one is not received the packet is retransmitted at a future random slot.

We use the following assumptions. The combined process of packet originations and packet retransmissions, from each set of terminals to a repeater, is Poisson. The probabilities of transmission by a repeater into different slots are independent. The probabilities of transmission by two or more repeaters into a randomly chosen slot are mutually independent; and the probabilities of transmission into a random slot by a terminal and by a repeater are independent. Furthermore, we assume that the terminal transmission range is short, so that it can reach only one repeater. On the other hand, the transmission from a repeater to the station can interfere with the transmission of terminals to I-1 other repeaters; $1 \leq I \leq m$.

The effect of directional antennas at repeaters is that the transmission from repeaters to the station is directed toward the station and does not interfere with the transmission of terminals to other repeaters. Thus, it is the special case with $I=1$. We notice, however, that directional antennas do not increase the capacity of the hop from repeaters to station because all antennas are directed toward the same physical location where the station is placed and where the conflicts may occur.

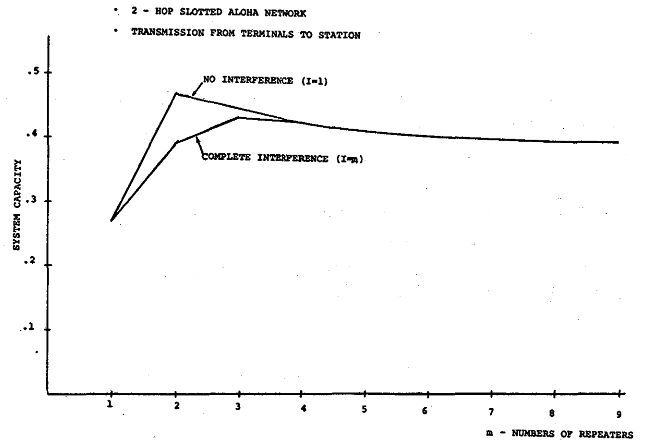


Figure 6—Network throughput vs. number of repeaters: Directional and non-directional antennas

Figure 6 shows the capacity of the system as a function of the number of repeaters, m, for $I=m$ and $I=1$, which is equivalent to omnidirectional and directional antennas respectively. One can see that there is a significant gain in capacity when using directional antennas only when $m=2$, and a small gain for $m=3$; for $m \geq 4$ the capacity of the system does not increase.

As far as the number of repeaters is concerned, one can see that 2 or 3 repeaters would be a good design; any additional repeaters that may be added because of other considerations (such as area coverage) will result in a reduction in the system capacity. Another problem investigated is the critical hop. That is, when the capacity of the system is reached, is it due to the saturation of the hop from terminals to repeaters or of the hop from repeaters to the station? The results demonstrate that when the number of repeaters, m, is small the critical hop is from terminals to repeaters, whereas when m is large the critical hop is from repeaters to the station. The exact number at which the change occurs depends on the interference parameter I.

Transmission from station to terminals

In this section, we consider the second channel which is used for transmission from the station to terminals via repeaters. It is assumed that the effective transmission range of the station is such that it interferes with the transmission from repeaters to terminals. However, we assume that terminals are not designed to directly receive from the station. We use the same assumptions as in the previous section. A transmission from the station to R_i can be interfered with

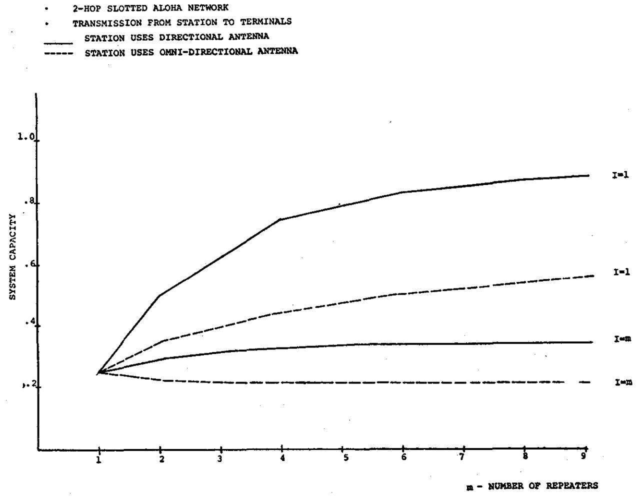


Figure 7—System capacity vs. number of repeaters for directional and non-directional antennas at the station and different interference parameters

by transmissions from the I repeaters in the interfering set of R_i when these repeaters transmit to their terminals (T's). A transmission from R_i to T can be interfered with by a transmission from the station to any repeater or by the $I-1$, excluding R_i, repeaters in the interfering set of R_i. For consistency with the interference model of the previous section we assume the same energy-per-bit-to-noise-density for detection with equal error rates, by the repeater and by the terminal and that the repeater uses a higher transmitter power than terminals.

Figure 7 shows the capacity of the system as a function of m for $I=1$ and $I=m$, both for an omnidirectional and directional antenna from the station to repeaters. Further investigations for this case were performed and the conclusions follow.

a. The interference of the station with the transmission of repeaters to terminals significantly reduces the system capacity. Thus, if possible, it is important to enable terminals to receive such transmissions directly, without retransmission by the repeater.

b. The system capacity is increased substantially when the interference level between repeaters is decreased. Note that this is not the case when transmitting to the station. Consequently, it is important to reduce the interference factor by a mechanism such as adaptive power.

c. A directional antenna at the station significantly increases the system capacity when the interference level between repeaters is low to moderate. This is not the case when the interference level is high, since the throughput on the hop from repeaters to terminals is limited due to this interference.

d. When the station has directional antennas, then multiple transmitters and antennas may further increase, significantly, system capacity. In this case, one can obtain a throughput greater than the capacity of a single channel.

ROUTING AND ACKNOWLEDGMENT CONSIDERATIONS

Routing considerations

There are two key objectives in developing a routing procedure for the packet radio system. First, we must assure, with high probability, that a message launched into the network from an arbitrary point will reach its destination. Second, we must guarantee that a *large number* of messages will be able to be transmitted through the network with a relatively small time delay. The first goal may be thought of as a *connectivity* or *reliability* issue, while the second is an *efficiency* consideration.

Neither of the above objectives can be accomplished in a network where the repeaters are unintelligent transponders. In such a network: (1) a single packet could saturate the capacity of the entire network if arbitrarily repeated; (2) packets could propagate indefinitely within the network; and (3) copies of the same packet could reach many different stations. Combinatorial analyses of message flow within these types of networks have shown the need for repeaters with sufficient computational capability to support more sophisticated routing and flow control procedures.[16]

In the next two subsections we describe two routing techniques. The first, "Undirected Routing," is useful for initializing the network, and for performing routing without the management or direction of a station. The second, "Directed Routing," requires the participation of the station but is much more efficient in two ways: (1) packets generally follow a single, shortest route to a station; and (2) paths are described by a hierarchical label which reduces the length of the necessary packet header.

Undirected routing

A rudimentary, but workable, routing technique to achieve connectivity at low traffic levels can be simply constructed by using a maximum handover number[17] and saving unique identifiers of packets at each repeater for a specified period of time. The handover number is used to guarantee that any packet cannot be indefinitely propagated in the net. Each time a packet is transmitted in the net, a handover number in the header is incremented by one. When the handover number reaches an assigned maximum, the packet is no longer repeated and that copy of the packet is dropped from the net. Thus, the packet is "aged" each time it is repeated until it reaches its destination or is dropped because of excessive age.

If the maximum handover number is set large, extensive artificial traffic may be generated in areas where there is a high density of repeaters. On the other hand, if it is set small, packets from remote areas may never arrive at stations. This problem can be resolved as follows: We assume that every repeater can calculate its approximate distance in number of hops to stations by observing response packets. The first repeater which received the packet from a terminal sets the maximum handover number based on its calculated distance from the station. The number is then decremented by one each time it is relayed through any other repeater. The packet is dropped when the number reduces to zero. When a station transmits a packet, it will set the maximum handover number by "knowing" the approximate radius in "repeaters" in its region.

Even if a packet is dropped after a large number of transmissions, local controls are needed to prevent packets from being successively "bounced" between two or a small number of repeaters which repeat everything they correctly receive. (Such a phenomenon is called "cycling" or "looping.") A simple mechanism to prevent this occurrence is for repeaters to store for a fixed period of time entire packets, headers, or even a field within the header that uniquely identifies a packet. A repeater would then compare the identifier of any received packet against the identifiers in storage at the repeater. If a match occurred, the associated packet would not be repeated.

The time allotted for storage of any packet identifier would

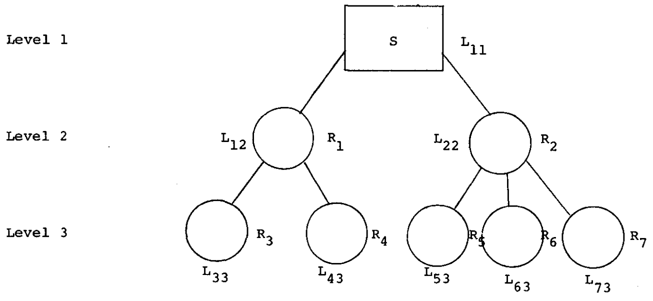


Figure 8

depend on the amount of available storage at a repeater and the number of bits required to uniquely identify the packet. For example, more than 4K packets could be uniquely identified with 12 bit words. Thus, 4K of storage could contain identifiers for more than 300 packets. With a 500 Kbps repeater to repeater common channel for broadcast and receive and 1,000 bit packets, this would be sufficient storage for over 1.5 seconds of transmission if the channel were used at full rate. Assuming a single hop would require about 20 milliseconds of transmission and retransmission time, a maximum hop number of 20 would guarantee that any packet would be dropped from the system because of an excessive number of retransmissions long before it could return to a previously used repeater not containing the packet identifier.

The combination of loop prevention and packet aging with otherwise indiscriminate repetition of packets by repeaters will enable a packet to travel, on every available path, a maximum distance away from its origin equal to its original handover number. Thus, if the maximum handover number is larger than the minimum number of hops between the terminal and the nearest station, a packet accepted into the net should reach its destination. Unfortunately, with this scheme, copies of the packet will also reach many other points, with each repetition occupying valuable channel capacity. However, if those packets for which adequate capacity is not available are prevented from entering the net, the network will appear highly reliable to accepted packets.

The above routing scheme is an *undirected*, completely distributed procedure. Each repeater is in total control of packets sent to it, and the stations play no active part in the system's routing decisions. (They must still play a role in flow control.) In the above procedure, no advantage is taken of the fact that most traffic is destined for a station, either as a terminus or as an intermediate point for communication with elements of a different network. Also, the superior speed and memory space of the station is ignored. For efficiency, one is therefore led to investigate *directed* (hierarchical) routing procedures.

Directed routing

A directed routing procedure utilizes the stations to periodically structure the network for efficient flow paths. Stations periodically transmit routing packets called *labels* to repeaters to form, functionally, a hierarchical point-to-point network, as shown in Figure 8. Each label includes the following information: (i) a specific address of the repeater for routing purposes, (ii) the minimum number of hops to the nearest station, and (iii) the specific addresses of *all* repeaters on a shortest path to the station. In particular, the label contains the address of the repeater to which a packet should preferably be transmitted when destined to the station.

When relaying a packet to its destination, the repeater addresses the packet to the next repeater along the preferred path. Only this addressed repeater will repeat the packet and only when this mechanism fails will other repeaters relay the message.

For simplicity, we describe routing for the case of a single station network. A label of repeater R_i of hierarchy level j will be denoted by L_{ij}; i, $j>1$. The station will have the label L_{11}. L°_{ij} will denote the label of the repeater which is the "nearest available" to the communicating terminal.

A label is composed of H subfields, where H is the maximum number of hierarchy levels ($H-1$ is the maximum number of hops on the shortest path between any repeater and the station). Every subfield has three possible entries, blank (BLK), a serial number (SER), or ALL. L_{ij} has j SER entries and $(H-j)$ BLK's as shown in Figure 9.

We say that L_{ij} "homes" on L_{kp}, $h(L_{ij})=L_{kp}$, if $p=j-1$ and the first $j-1$ subfields of both are identical. If two repeaters at level j home on the same repeater, their labels will differ only in the entry to subfield j.

As an example, if we use 3 bits per subfield, the labels of the station and the repeaters of the network shown in Figure 8 are as follows:

	Subfield 1	Subfield 2	Subfield 3
L_{11}	0 0 1	0 0 0	0 0 0
L_{12}	0 0 1	0 0 1	0 0 0
L_{22}	0 0 1	0 1 0	0 0 0
L_{33}	0 0 1	0 0 1	0 0 1
L_{43}	0 0 1	0 0 1	0 1 0
L_{53}	0 0 1	0 1 0	0 0 1
L_{63}	0 0 1	0 1 0	0 1 0
L_{73}	0 0 1	0 1 0	0 1 1

In this example, a subfield in which all bits are "0" is considered "blank." Note that all entries in Subfield 1 are the same since all repeaters home (eventually) on the same station.

The packet header, shown in Figure 10 includes the following routing information. L_{kn} is the label of the repeater to

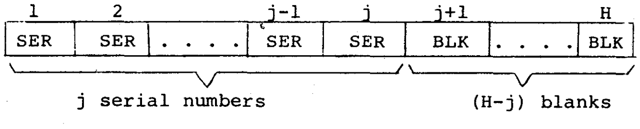


Figure 9

which the packet is currently addressed. The complete packet will *always* be transmitted to a *specific device*; other devices which may receive the packet will drop it. The shortest path from a terminal to the station consists of L°_{ij}, $h(L^\circ_{ij})$, $h(h(L^\circ_{ij}))$, up to L_{11}, in the given order, and in the reverse order when routing from station to terminal. When a specific repeater along the shortest path is not known (by the terminal) or not available, then the terminal or repeater (which has the packet) will transmit *only the header part* of the packet, trying to identify a specific repeater. In that case, the label L_{kn} will include some entries ALL. To see how the proposed routing technique would operate, we trace the sequence of steps performed when a terminal attempts to transmit a packet to the station.

When a previously silent terminal begins to communicate, it first identifies a repeater or a station in its area. It transmits only the header part of the packet with all entries in L_{kn} set to ALL. The header is addressed to all repeaters and stations that can hear the terminal. A device which correctly receives this header substitutes its label in the space L_{kn} and repeats the header. This particular L_{kn} is also L°_{kn} and will be used by the terminal to transmit all packets during this period of communication. If a terminal is stationary, it can store this label for future transmissions. L°_{kn} begins to transmit the complete packet along the shortest path to the station.

Suppose that L_{ij} along the shortest path is not successful in transmitting the packet to $h(L_{ij})$. Then L_{ij} begins the search stage of trying to identify another repeater. In the first step, it tries to identify a repeater which is in level $p \leq j-1$. This is done by using the label shown in Figure 11. The header is addressed to all repeaters in levels 2 to $j-1$, which eventually home on L_{11}. If this step is not successful, in the second (last) step, L_{ij} tries to identify any available repeater by using a label in which the first entry is SER and all other entries are ALL. When a specific repeater is identified and receives the packet, it transmits the packet on the shortest path from its location.

Note that if repeaters have sufficient storage, they can save alternative labels and thus reduce the necessity of searching for a specific repeater. Alternative solutions in which repeaters have multiple labels are also possible.

Acknowledgment considerations

Acknowledgment procedures are necessary both as a guarantee that packets are not lost within the net and as a flow control mechanism to prevent retransmissions of packets from entering the net. Two types of acknowledgments are common in packet oriented systems:

L_{kn}	L°_{ij}	OTHER HEADERS AND PACKET INFORMATION
TO	LABEL OF NEAREST REPEATER TO THE TERMINAL	

Figure 10

1	2	3		j-1	j	j+1		
SER	ALL	ALL	. . .	ALL	BLK	BLK	. . .	BLK

Figure 11

1. Hop-by-Hop Acknowledgments (HBH Acks) are transmitted whenever a packet is received successfully by the next node on the transmission path.
2. End-to-End Acknowledgments (ETE Acks) are transmitted whenever a packet correctly reaches its final destination within the network.

In a point-to-point oriented network such as the ARPANET, HBH Acks are used to transfer responsibility (and thus open buffer space) for the packet from the transmitting node to the receiving node. This Ack insures prompt retransmission should parity errors or relay IMP buffer congestion occur. The ETE Ack serves as a flow regulator between source and destination and as a signal to the sending node that the final destination node has correctly received the message. Thus, the message may be dropped from storage at its origin.

Both types of Ack's serve to ensure message integrity and reliability. If there is a high probability of error free transmission per hop and the nodes have sufficient storage, the Hop-by-Hop scheme is not needed for the above purpose. Without an HBH Ack scheme, one would retransmit the packet from its origin after a time out period expired. One introduces the HBH Ack to decrease the delay caused by retransmissions at the expense of added overhead for acknowledgments.

In the packet radio system, the overhead can be kept small by listening, whenever possible, for the next repetition of the packet on the common channel instead of generating a separate acknowledgment packet.

The value of an End-to-End acknowledgment is sufficiently great that it can be assumed present *a priori*. However, the additional use of a Hop-by-Hop acknowledgment is not as clear. Therefore, in this section, we examine the question of whether the ETE Ack is sufficient, or whether one needs a Hop-by-Hop (HBH) acknowledgment in addition. The problem is therefore whether a HBH Ack is superior to an ETE Ack with respect to throughput and delay, since the ETE Ack ensures message integrity. It is noted that the routing and flow control by devices in the network depend on the type of acknowledgment scheme used.

We consider a simple case where $(n-1)$ repeaters separate the packet radio terminal from the destination station. As-

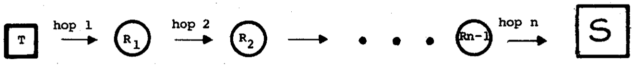


Figure 12

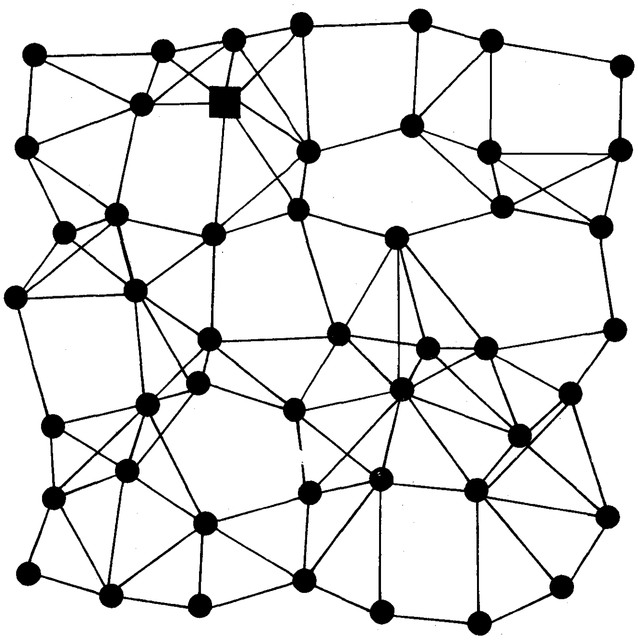


Figure 13—Connectivity of repeaters and stations

suming that the terminal is at a distance of "one hop" from the first repeater, one obtains the n-hop system shown in Figure 12.

A simple model is used to evaluate the total average delay that a packet encounters in the n-hop system when using HBH and ETE acknowledgment schemes. When the ETE acknowledgment scheme is used, every repeater transmits the packet a single time. If the packet does not reach the station, retransmission is originated by the terminal. The ETE acknowledgment is sent from the station. In the HBH scheme, repeaters store and retransmit the packet until positively acknowledged from the next repeater stage.

If, after a terminal (or a repeater in the HBH case) transmits the packet, an acknowledgment does not arrive within a specified period of time, it retransmits the packet. The waiting period is composed of the time for the acknowledgment to arrive when no conflicts occur plus a random time for avoiding repeated conflicts.

Two different schemes for ETE acknowledgment and one scheme for HBH acknowledgment have been studied. Curves for the total average delay as a function of the number of hops and the probability of successful transmission per hop were obtained. Two cases were considered: One in which the probability of success is constant along the path and another in which the probability of success decreases linearly as the packet approaches the station. This model is consistent with Packet Radio Network, because all packets which originate from terminals are routed toward a station. Finally, channel utilizations are compared when using ALOHA[18,19] random access modes of operation.

The results show that the HBH scheme is superior in terms of delay and channel utilization. The difference in performance becomes significant when the number of hops increases or when the probability of successful transmission per hop is relatively low as is the case in the Packet Radio System. For example, in a five hop system, if the probability of success per hop is 0.7, then the total average delay is 12.5 and 53 packet transmission times for the HBH and ETE acknowledgment schemes, respectively.

RANGE, POWER, DATA RATE AND CAPACITY CONSIDERATIONS

Transmission range and network interference

A variety of situations are possible concerning the range and interference patterns of devices. For example, with identical *r.f.* elements and similar antenna placements, Repeater to Repeater range is the same as Terminal to Repeater range. This need not always be the case since repeaters can often have higher antennas than terminals, (especially hand held terminals). Thus, if repeaters are allocated for *area coverage of terminals*, the repeater range will be higher than terminal range and higher network connectivity or device interference will result.

The problem which then arises is to determine the impact of this interference on system performance. Alternatively, one may seek to reduce repeater transmission power when transmitting in the repeater-station network. As an indication of the tradeoffs that occur, two systems with common channels and single date rates (CCSDR) were simulated, one with high interference CCSDR (HI), and the other with low interference CCSDR (LI). As a first step, the routing labels of the two systems were the same and are shown in

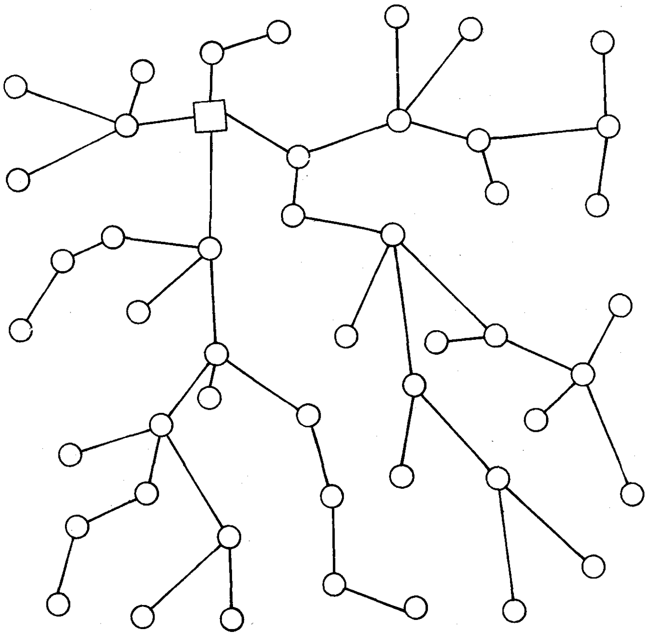


Figure 14—Hierarchical labeling scheme for low interference model

TABLE I—Performance Measures

	OFFERED RATE [%]	THROUGH-PUT [%]	DELAY OF IP *TO STATION [Terminal Slots]	RATE OF STATION RESPONSE	PROB. STATION BUSY	% OF IP* BLOCKED	TOTAL % OF IP* LOSS	TERMINALS REMAINING
CCSDR (LI)	13	5.95	40.11	1.14	.53	2.98	32.83	13
CCSDR (HI)	13	10.55	23.93	1.81	.43	9.83	9.83	13
CCSDR (HI) (Improved Labels)	13	12.14	16.61	2.06	.50	10.63	11.41	10

* Information Packet

Figure 14. The interference of the CCSDR (LI) system is shown in Figure 13 and the interference of the CCSDR (HI) system in Figure 15. (Figure 15 shows only the connectivity for two devices in the network.) A different label assignment for the high interference system is shown in Figure 16.

The results are shown in Figure 17 and Table I. Figure 17 shows the throughput of the two systems as a function of time while Table I summarized other measures of performance. The third row of Table I summarizes performance of the high interference system under an improved set of repeater labels. It is clear that the high interference system has better performance than the low interference systems. The only measure of the low interference system which is better is terminal blocking which is a direct result of the low interference feature. In fact, CCSDR (LI) is saturated at the offered traffic rate. This can be seen from the fact that the throughput is decreasing as a function of time; the relatively high total loss; and the low station response.* The CCSDR (HI) with improved labels compared in Table I, has better performance than the other two systems. This indicates the importance of proper labeling. The experiments of this section suggest that it is preferable to use high transmitter power to obtain long repeater range, despite the network interference that results.

Single versus dual data rate signalling

Because of the problem of developing different hardware (r.f. and digital) for repeaters, terminals and stations, a basic

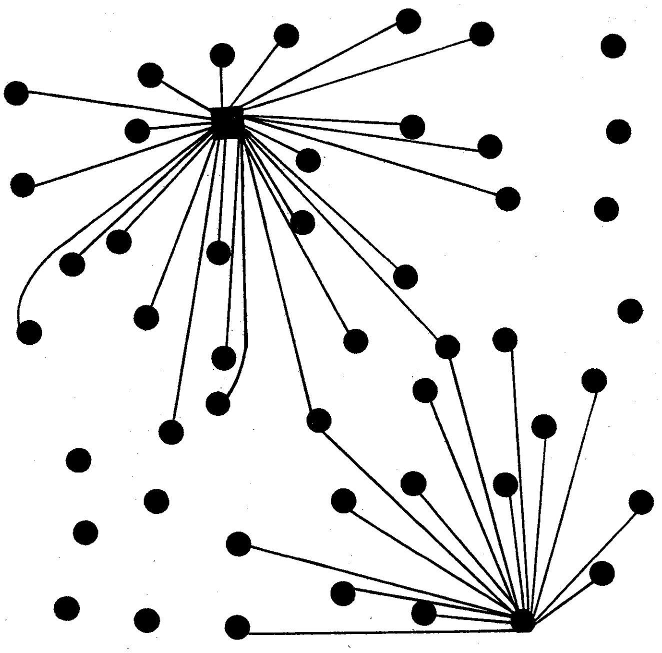


Figure 15—Interference of CCSDR (HI) system

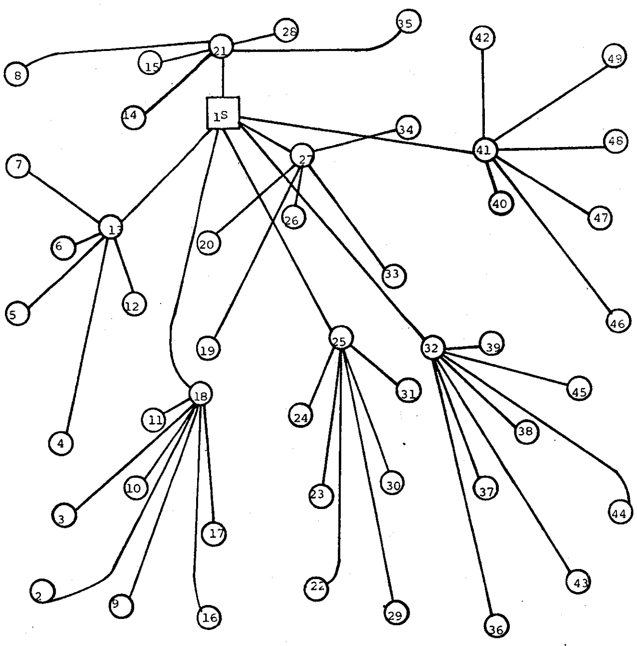


Figure 16—CCSDR (HI) system with improved labeling

* The average number of station response packets assumed for these studies is 2.0.

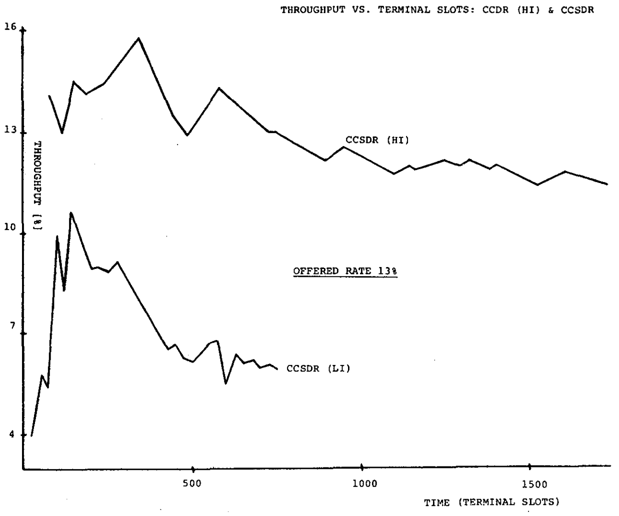


Figure 17—Throughput vs. terminal slots: CCDR (HI) and CCSDR (LI)

design decision for the experimental system was to use the same hardware, called a *Packet Radio Unit* (PRU) for all three devices. Because of this decision, these devices have the same transmission capabilities. However, in many applications the antennas of repeaters and stations can be placed in elevated areas, and therefore, the quality of the transmission links between devices within this category can be superior to the quality of the links between terminals and repeaters or terminals and stations. Consequently, an important problem is to determine whether repeaters and stations should use this extra capability to achieve longer range or higher data rates than the terminals.

This section discusses some of the studies performed to investigate this tradeoff. The results indicate that a dual data rate approach can significantly increase system performance. Thus, the Packet Radio Unit is being implemented with this feature.

Among the systems studied were:

- The common channel and single data rate system CCSDR (HI) of the previous section with improved labels to take advantage of the high range to improve the routing labels of repeaters and obtain fewer hierarchy levels which we denote by CCSDR. The routing labels used are shown in Figure 16, and the connectivity is shown in Figure 15.
- A Common Channel Two Data Rate (CCTDR) system with the routing labels as in Figure 14 and connectivity as in Figure 13.

In the CCTDR system, the terminal has a low data rate channel, the same rate as in the single data rate system, for communication with a repeater or station. Repeaters and station have two data rates. The high data rate is used for communication in the repeater-station network. The two data rates use the same carrier frequency so that only one can be used at a time.

The two systems were tested with offered rates of 13 percent and 25 percent.* The throughputs as a function of time for the two runs are shown in Figures 18 and 19, respectively; and the summary of other measures is given in Table II. The comparison demonstrates that the CCTRD system is superior to the CCSDR system, in terms of throughput, delay, and other measures. One can see that the CCSDR system is saturated at an offered rate of about 13 percent.

- *Effect on Blocking Level*

In Table II, one can see that one reason for the relatively low throughput of the CCSDR system at an offered rate of 25 percent is due to blocking. Furthermore, the fraction of time that the station is busy has decreased. This may suggest

* In the simulation runs we used the inverse square law for the relation between data rate and distance, rather than the result in 20; this however, favors CCSDR.

TABLE II—Performance Measures

	OFFERED RATE [%]	THROUGH-PUT [%]	DELAY OF IP *TO STATION [Terminal Slots]	RATE OF STATION RESPONSE	PROB. STATION BUSY	% OF IP* BLOCKED	TOTAL % OF IP* LOSS	TERMINALS REMAINING
CCSDR	13	12.14	16.61	2.06	.50	10.63	11.41	10
	25	12.20	34.97	1.61	.48	29.50	32.95	23
CCTDR	13	12.39	4.91	1.99	.26	1.59	1.59	9
	25	23.33	11.51	1.97	.31	3.31	3.31	34

* Information Packet

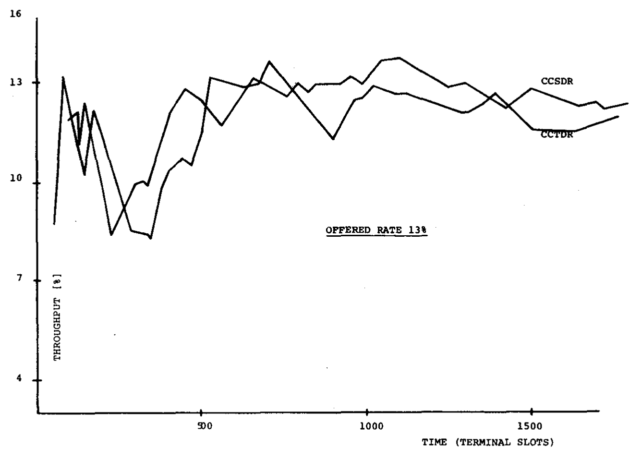


Figure 18—Throughput vs. terminal slots

that the station may be able to handle more terminals providing they are able to enter the system. To examine this point, we ran the CCSDR system with offered rate of 25 percent, and relaxed the constraint for entering the system. Rather than resulting in better performance, this step resulted in reduction in blocking and increase in delay. The throughput increased to 12.63 percent, the blocking decreased to 18.35 percent and the total loss decreased to 30.73 percent. On the other hand, the delay increased to 57.82, the fraction of time the station is busy increased to .57, and the rate of station response decreased to 1.32.

To conclude, when we enabled more terminals to enter the system, the throughput increased insignificantly, from 12.20 percent to 12.63 percent; on the other hand, the average packet delay *increased significantly*, from 34.97 to 57.82 terminal slots. This suggests that one of the important design problems in the packet radio system is the blocking level of terminals.

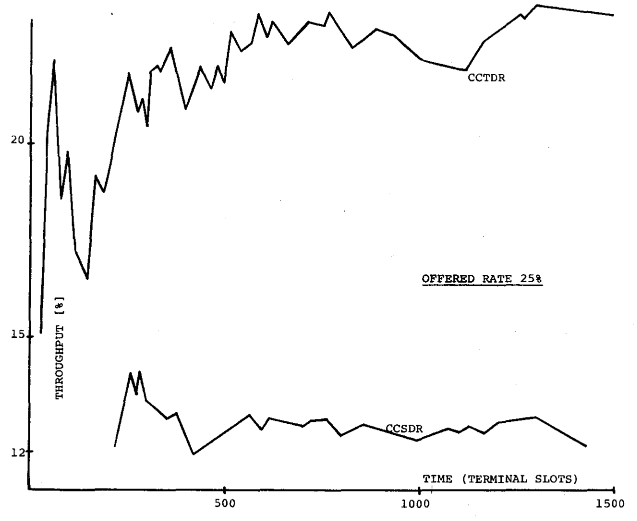


Figure 19—Throughput vs. terminal slots

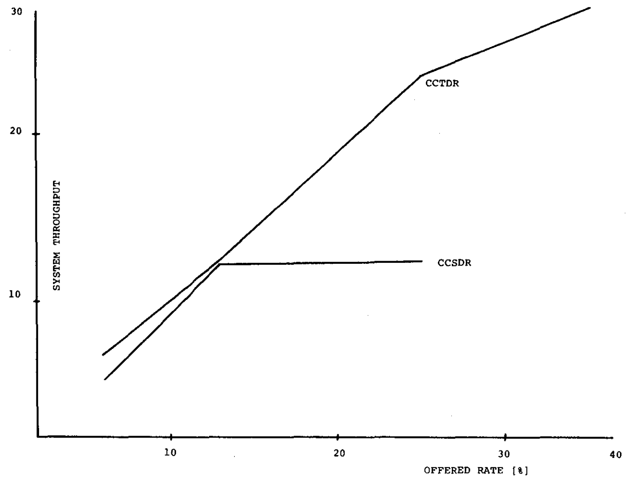


Figure 20—System throughput vs. offered rate

Throughput, loss, and delay of CCSDR and CCTDR systems

Similar to curves of throughput versus channel traffic obtained analytically,[2] we can attempt to draw curves of system throughput vs. offered rate for estimating the maximum throughput by simulation. Figure 20 shows the throughput versus offered rate for CCSDR and CCTDR systems. The curves are linear for low offered rates and saturate when the offered rate increases.

For the CCSDR system one can see that the throughput is practically the same when the offered rate is increased from 13 percent to 25 percent. This and the other measures (see Table II), (for example, the rate of station response) show that the system is overloaded at a 25 percent offered rate. On the other hand, the system seems to operate at steady state at an offered rate of 13 percent (rate of station response 2.06). A rough estimate of maximum throughput for this system would be between 12 percent and 15 percent. Similar observations of the performance measures led to an "estimate" of between 27 percent and 30 percent for the maximum

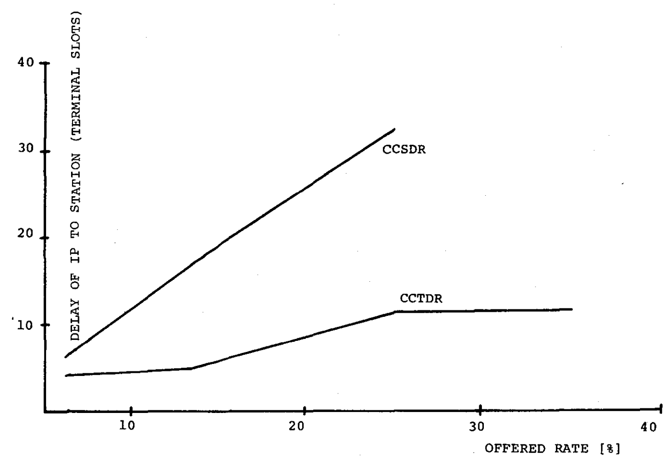


Figure 21—Terminal-station delay vs. offered rate

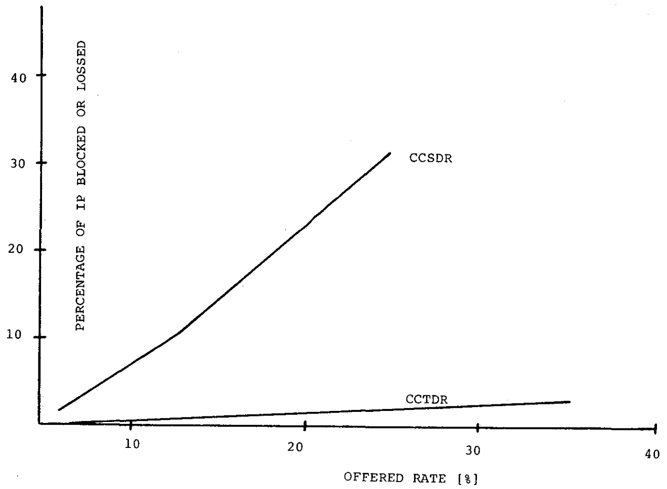


Figure 22—IP blocking vs. offered rate

throughput of the CCTDR system in the specified repeater configuration.

The average delay of the first Information Packet from terminal to station, and the Total Loss, as a function of offered rate are shown in Figure 21 and Figure 22, respectively.

Remark: There are many parameters in the simulation program which we have not experimented with and which affect the quantities discussed above. One parameter, α, which is significant in determining the maximum throughput is the ratio of the average number of packets from station to terminal to the average number of packets from terminal to station. The effect of this parameter has been analyzed in Reference 14 for a slotted ALOHA random access mode. It has been shown that the maximum throughput is increased in the Common Channel system when α increases, and the maximum throughput tends to 100 percent of the data rate when α tends to infinity. We expect that this parameter has a similar effect for the mode of access simulated. In the results reported here α is 2.0 which is small compared to usual estimates for terminals interacting with computers. Thus the estimates of this section will probably somewhat underestimate the maximum throughput.

CONCLUSIONS

As is evident, packet radio offers a new and challenging area for network analysis and design. These studies have merely touched upon crucial areas. Further studies, to develop methodology, to provide support for hardware and software design, and to effectively control and manage network resources are currently under way. These studies will:

a. Estimate system capacity as a function of terminal-repeater and repeater-repeater signaling rates for multistation networks.
b. Compare the performance of systems with varying degrees of receiver capture, multiple channels, directional antennas and multiple detectors.
c. Determine efficient operating parameters including time out intervals, handover numbers, and number of retransmissions.
d. Determine relationship between number of repeaters, throughput, delay and blocking.
e. Compare the efficiency of direct terminal to terminal routing versus hierarchical routing in multistation networks.
f. Estimate throughput, delay and blocking for multistation networks.
g. Develop and test multistation algorithms for routing, labeling, and relabeling.
h. Develop high level global flow control algorithms to allow effective utilization of system resources.
i. Determine network control strategies to identify and monitor network congestion and eliminate failure conditions.
j. Formulate dynamic reliability and survivability criteria and develop algorithms for network reliability analysis and design.
k. Develop algorithms for configuring packet radio networks to meet specified reliability and survivability criteria.

REFERENCES

1. Kahn, R. E., "The Organization of Computer Resources Into a Packet Radio Network," *National Computer Conference*, May, 1975.
2. Kleinrock, L. and F. Tobagi, "Random Access Techniques For Packet Radio Networks," *National Computer Conference*, May, 1975.
3. Turin, G. L., "A Statistical Model of Urban Multipath Propagation," *IEEE Transactions on Vehicular Technology*, Vol. VT-21, February, 1972, pp. 1-9.
4. Frisch, I. T., "Technical Problems in Nationwide Networking and Interconnection," *IEEE Transactions on Communications*, January, 1975.
5. Garrett, J. and S. Fralick, "A Technology For Packet Radio," *National Computer Conference*, May, 1975.
6. Fralick, S., D. Brandin, and F. Kuo, Harrison, "Digital Portable Terminals," *National Computer Conference*, May, 1975.
7. Burchfiel, J., R. Tomlinson, and M. Beeler, "Functions and Structure of a Packet Radio Station," *National Computer Conference*, May, 1975.
8. Abramson, N., R. Binder, F. Kuo, Wax, D. Oklinaka, "ALOHA Packet Broadcasting—A Retrospect," *National Computer Conference*, May 1975.
9. Okamura, Y. et al., "Field Strength and Its Variability on VHF and UHF Land-Mobile Radio Service," *Review of the Electrical Communication*, Vol. 16, No. 9-10, September-October, 1968.
10. Van Slyke, R. and H. Frank, "Network Reliability Analysis-I," *Networks*, Vol. 1, No. 1, No. 3, 1972, pp. 279-290.
11. Garfinkel, R. and G. Nemhauser, *Integer Programming*, J. Wiley, New York, 1972.
12. Karp, R., "Reducibility Among Combinatorial Problems," *Complexity of Computer Computations*, R. Miller and J. Thatcher (eds.), Plenum, 1972.
13. (Network Analysis Corporation), "The Practical Impact of Recent Computer Advances on the Analysis and Design of Large Scale Networks," *Third Semiannual Technical Report*, June, 1974, available from Defense Documentation Center, Arlington, Va.

14. Gitman, I., R. M. Van Slyke and H. Frank, "On Splitting Random Access Broadcast Communication Channels," *Proceedings of the Seventh Hawaii International Conference on System Sciences*, Subconference on Computer Nets, January, 1974.
15. Gitman, I., "On the Capacity of Slotted ALHOA Networks and Some Design Problems," *IEEE Transactions on Communications*, March, 1975.
16. NAC (Network Analysis Corporation), "The Practical Impact of Recent Computer Advances on the Analysis and Design of Large Scale Networks," *Second Semiannual Technical Report*, December, 1973.
17. Boehm, S. P. and P. Baran, "Digital Simulation of Hot-Potato Routing in a Broadband Distribution Communications Network," Rand Corporation, *Memorandum RM-3103-PR*, August, 1964.
18. Abramson, N., "The ALOHA System—Another Alternative for Computer Communication," *AFIPS Conference Proceedings*, Vol. 37, November, 1970, pp. 281-285.
19. Abramson, N., "Packet-Switching with Satellites," *National Computer Conference*, June 1973, pp. 695-702.
20. Fralick, S. C., "R. F. Channel Capacity Considerations," available from *ARPA Network Information Center*, Stanford Research Institute, Menlo Park, Calif., 1974.

Technological considerations for packet radio networks*

by STANLEY C. FRALICK
Stanford Research Institute
Menlo Park, California

and

JAMES C. GARRETT
Rockwell International
Richardson, Texas

INTRODUCTION

The application of packet-switching techniques to radio channels has provided a solution to many computer-communications problems previously unsolved.[1] For example, a packet radio network can readily be designed to provide area coverage at data rates fast enough to support interactive operations for thousands of users having a variety of terminals such as hand-held devices, TTY-like devices, display devices, computers, and unattended sensors. Since the interconnections are by radio, the users can be fixed or mobile, and the network can be easily moved. Furthermore, it can be readily established in remote or primitive areas where a wired network would be impossible, and total connectivity of users will be provided.

Packet radio provides an economical solution to these needs when the user demand for information transfer has a high peak to average ratio or a "bursty" behavior.[2] This is, in fact, the case for a large number of data communication users. Packet switching, as opposed to circuit switching, allows resources, including the transmission capacity, to be dynamically allocated to satisfy data transfer demands.

A typical packet radio network uses radio broadcast properties over one common channel to provide area coverage. The network traffic is quantized into discrete segments called packets and a header is attached that defines the routing protocol of the packet.[3]

Packets are independently transferred from one network element to another asynchronously with control mechanisms for the management of these transactions distributed throughout the network. A typical packet radio network consists of three primary functional elements: Terminal, Station, and Repeater. A typical network is shown in Figure 1. The terminal[4] is the user's interface to the network. The station[5] has the responsibility for over-all management of the network including initialization, routing, flow control, directory, and accounting functions. It also serves as the gateway from the network to other networks. In a network covering a small area, terminals and a station suffice.[2] However, terminals must have limited power to be portable, and this power limits their range. To provide coverage over extended areas, repeaters are needed. The repeater has the function of extending the range of station-terminal links and providing a mechanism for distributing the network management logic. It, therefore, receives and retransmits packets with the additional responsibilities of detecting errors and invoking routing protocols dictated by the station.

A repeater element contains a radio section which provides access to the network radio channel and a digital section which performs the logical functions of error control and packet routing. Figure 2 illustrates this organization of the repeater element.

An experimental repeater has been constructed by Collins Radio Group and is undergoing tests at Stanford Research Institute. A description of this experimental packet radio repeater along with a discussion of the physical limitations of the RF channel and design constraints of modern technology on this development are presented.

MULTIPATH CHANNELS

The experimental packet radio repeater operates in the 1 to 2-GHz range. The ALOHA network[2] operates at 400 MHz. Potential networks might be designed almost anywhere in the UHF band. Such networks will be required to operate in urban areas with high-rise buildings, as well as in suburban and rural areas such as jungles and deserts. Propagation in these topologies is highly variable, and significant multipath results.

An example of multipath propagation is shown in Figure 3. The pulse distortion here is very apparent. This picture was taken in Palo Alto at 1370 MHz using a spread-spectrum transmitter and matched filter receiver giving a 200 ns time-resolution capability.

* This work was supported by the Advanced Research Projects Agency on Contract No. DAHC 15-73-C-0187 and Contract No. DAHC 15-73-C-0192.

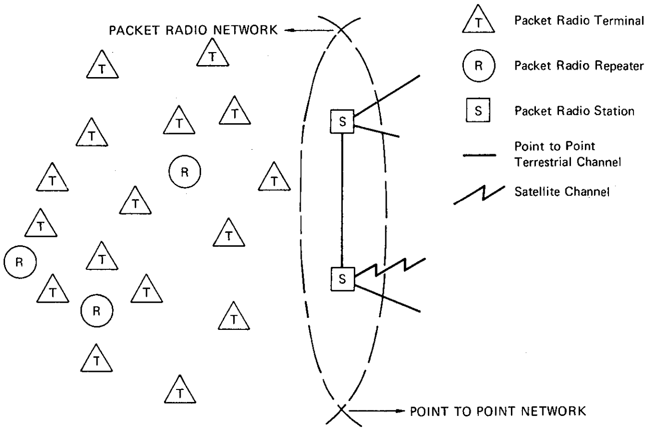


Figure 1—A packet radio network

The signal distortion introduced by the multipath channel places a limit on the bit-rate which the channel will support. The delay spread of the channel is a measure of the distortion introduced. It is determined by the 3-dB points on the channel impulse response, as shown in Figure 3. If this spread is greater than the duration of one signaling element, which is usually a single bit, then intersymbol interference will result, and errors will occur even at high signal-to-noise ratios.

RANDOM PATH LOSS

Satisfactory performance of the rf links of a network depend upon the receipt of adequate signal power to overcome the noise at the receiver.

Because coverage is so terrain dependent, it is possible only to present "typical" range curves. Such curves are shown in Figure 4 for a 10-W transmitter and a receiver with an 8-dB noise figure and 200-kHz noise bandwidth, assuming no nonthermal noise components. These curves were calculated using Okumura's curves[6] of median signal strength corrected for terrain type and antenna height.

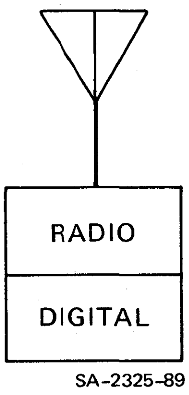


Figure 2—Basic organization of packet radio repeater

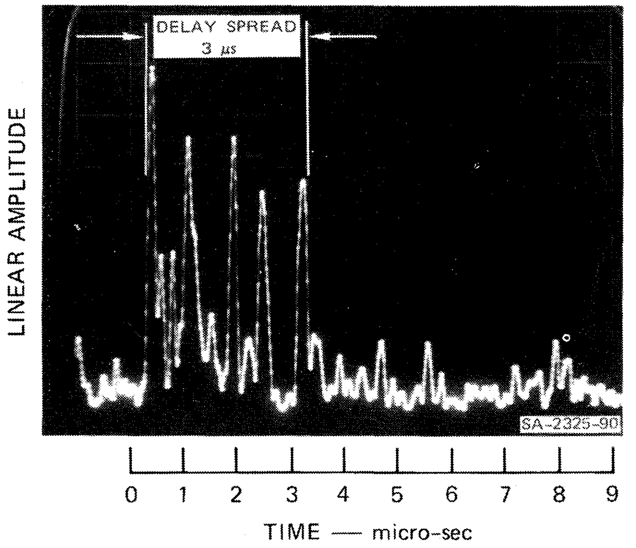


Figure 3—Channel impulse response in suburban area

Notice that the signal strength is 40 to 80 dB less than might be calculated using a free-space propagation model, and that the median levels vary as $1/R^{\alpha}$ where α is between 3 and 4. This very rapid decrease in signal strength with range means that a low-powered terminal cannot provide adequate area coverage in many cases, so

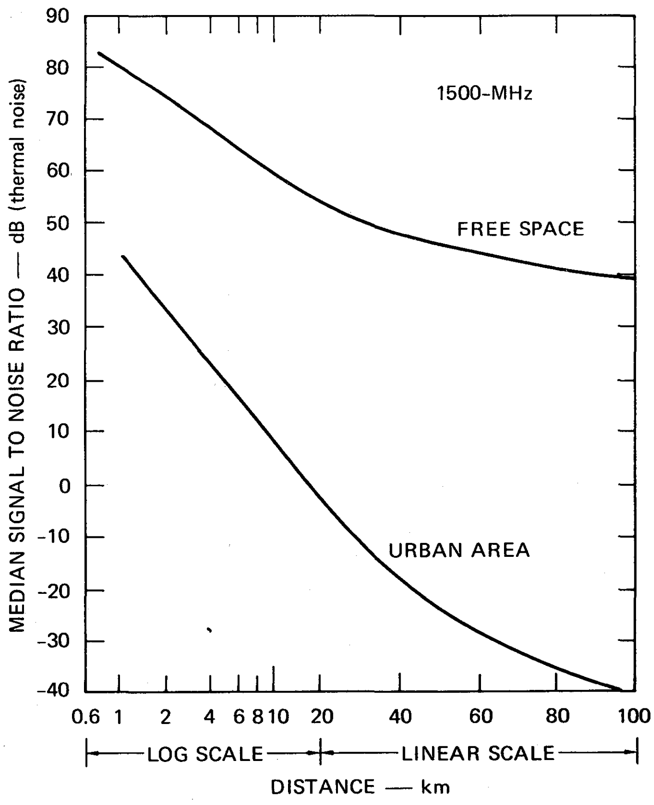


Figure 4—Communication link range

that either terminals must be very high-powered or some range-extension means must be found.

For example, suppose that a modulation scheme was used which required 13-dB signal-to-noise ratio for satisfactory performance. Then to obtain a 20-km range in an urban area at a 100-kBs bit rate, the transmitter power required would exceed 10 kW. Clearly this is excessive for a small portable terminal, and networks to support such terminals must have rf repeaters spaced closely enough so that terminal-repeater range never exceeds the capability of the terminal transmitters.

Notice that these curves ignore the effects of non-thermal noise. If the system must maintain a positive signal-to-noise ratio even in the presence of impulsive (usually man-made) noise, the transmitter power to provide a given range must be increased in proportion to the energy in the noise impulses. As discussed below, this often exceeds 40 dB.

NOISE

There are at least three major types of noise in this band. These are characterized by their nature as background (or Gaussian), impulsive (or non-Gaussian), and interference. The latter will be considered separately.

Link power budget and repeater LOS coverage calculations are based on the background noise level. In the UHF band, this noise is primarily of thermal origin and may be taken as thermal noise at 20° C. Such noise is assumed to have uniform spectral density of -172 dBm/Hz and to have a Gaussian amplitude distribution. Receiver noise figures in this band will increase the background noise level from 4 to 8 dB.

The primary impulsive noise in the UHF band appears to be derived from automobile ignition spark discharges. Hence, it varies with population density. Other sources of impulsive noise are also man-made, such as arc-welders, electric trains, and ac power distribution systems. All are population density dependent.

Impulsive noise is not generated at a high level, so that it is only effective at short range. For example, a terminal on a main city street, or near a freeway might experience noise impulses exceeding the thermal noise by 60 to 80 dB; but a repeater, several hundred feet above the street on a tall building, might experience impulses only 30 to 40 dB above thermal noise, and a repeater on a mountain in a suburban or rural area might experience only background noise.

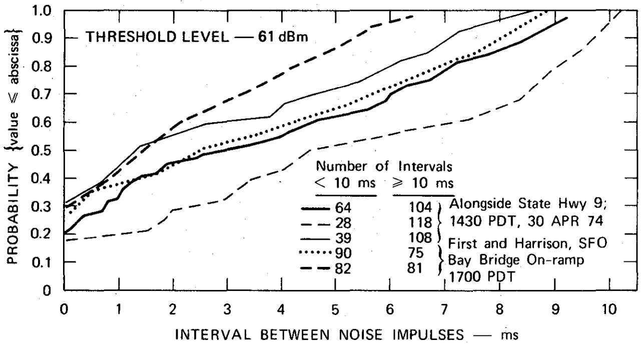


Figure 5—Cumulative distribution of impulse noise threshold crossings

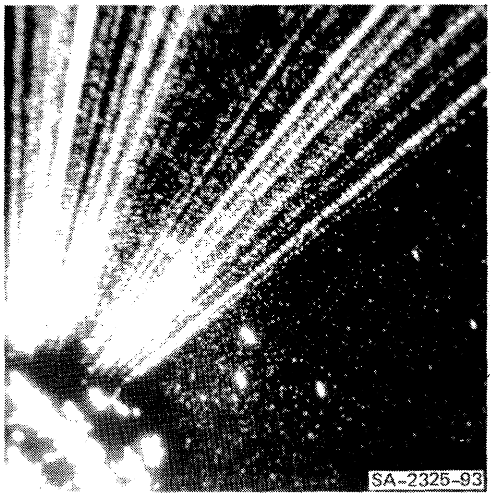


Figure 6—Effect of one-second burst of signal on radar plan-position indicator

It is not desirable to limit repeater coverage so that all impulses can be overpowered by the signal. This would require excessive transmitter power in worst-case situations. Instead, because an impulse may affect only a few bits per packet, provision has been made to implement and experiment with error-correcting coding techniques. Such techniques will depend on the detailed time-statistics of the noise impulses which have been collected and are discussed in Reference 7.

The importance of the impact of impulsive noise on the design of a packet radio network can be partially appreciated from an examination of Figure 5, which shows the cumulative probability distribution of noise impulse threshold-crossing intervals measured at several urban locations. This figure shows that noise impulses exceeding the background thermal noise by 46 dB arrive with intervals less than 5 ms, 50 percent of the time, and with intervals less than 10 ms almost all of the time. Thus packets which are longer than 10 ms will almost always experience at least one very large noise pulse. Since it is impractical to design the system to overpower these impulses, it is necessary to reduce packet lengths, so that only a small proportion will experience noise pulses, and to provide error detection. Longer packets in urban areas must include error correction or the channel throughput will be very low.

INTERFERENCE

The interference problem has two aspects. The system must avoid interference from other users, and it must avoid interfering with other users. The specific nature of the problem depends on the specific community of users

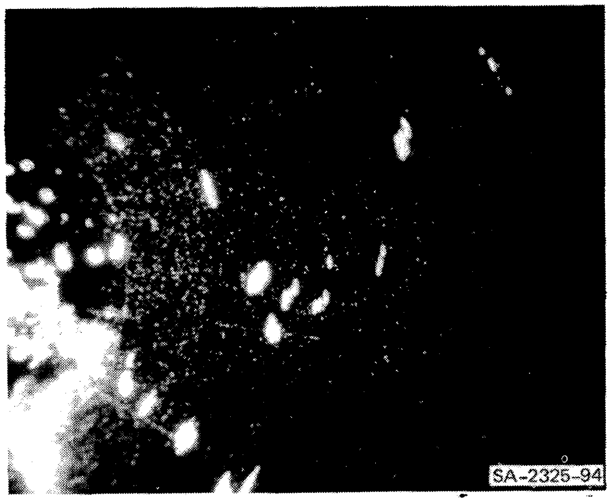


Figure 7—Effect of reducing burst duration to one millisecond

coexisting. Preliminary coexistence experiments have been performed which suggest that it is possible to coexist with even such difficult users as radars if the repeater coverage is sufficiently small and error-correcting codes are used. Coexistence in other user communities should be much simpler and exploration of this issue will be a major goal of future experiments.

Figure 6 is a picture of one segment of a radar plan-position indicator showing the effect of a one-second burst from a signal similar to that planned for the first experimental network using a 6-W transmitter at a range of 8 miles. The very small dots are caused by noise, the larger dots are aircraft, and the large strobes are the packet radio signal. Figure 7 shows the effect of a full power burst of signal for 1 ms. and Figure 8 shows the effect of a one-half second burst with power reduced by 20 dB. For these tests the signal was not tuned to the center of the radar receiver passband. The two frequencies differed by 13 MHz, or three times the nominal radar bandwidth.

ACHIEVABLE RANGE

The transmitted power, modulation, coding scheme, and bit rate of packet radio equipment will interact with the radio channel in a complex way to limit the range of a repeater-terminal link. It is this limitation which furnishes the fundamental motivation for a repeater-oriented network philosophy. The primary need for a packet radio network is to service portable terminals which are small, (someday hand-held) light weight and conveniently used. If the terminals are not small and light weight, they will not be portable and many of the advantages of a radio connection will be lost. Because of transmitter power constraints, limited terminal size implies limited terminal range, so that repeaters must be used to extend area coverage.

The desired size and weight restrictions on packet radio equipment imply both an average and a peak power constraint. The average power constraint has little effect on range, since the transmitter duty cycle for packet-switched terminals is so low that the average transmitter power is only a few percent of the total terminal or repeater power consumption. On the other hand, the peak power constraint limits the power transmitted during a burst, and therefore the range. This peak power constraint comes about because the typical burst exceeds the thermal time-constant for semiconductor devices used as rf power amplifiers; i.e., the burst lasts long enough to overheat the rf power amplifier if the peak burst-power exceeds the average-power rating of the device. This can be overcome by paralleling amplifiers or otherwise increasing device bulk—only at the cost of increasing packet radio equipment bulk.

The rf channel distorts transmitted signals and adds noise, so that the performance of an rf communication link will typically be less reliable than a wire link. Reliability is measured in terms of the probability of a bit error (P_{BE}). If the rf channel simply adds white gaussian noise, then P_{BE} will depend monotonically on the ratio of received energy in one bit (E_b) to background noise intensity (N_0). Since the packet radio transmitter will be peak-power limited, the received energy will be proportional to the product of peak power (P) and bit duration (T). Assuming a worst-case urban propagation condition in which the received energy varies inversely with the fourth power of range (R) (See Figure 4), the performance will be

$$P_{BE} = f\left(\frac{PT}{N_0R^4}\right)$$

where $f(\cdot)$ is a monotonic decreasing function depending on the type of signalling scheme used. The tradeoff relation between T and R is shown in Figure 9 for three types of modulation and coding schemes, assuming that a bit error probability of 10^{-5} must be achieved and that the transmitter operates at 1500 MHz with 10 W peak power

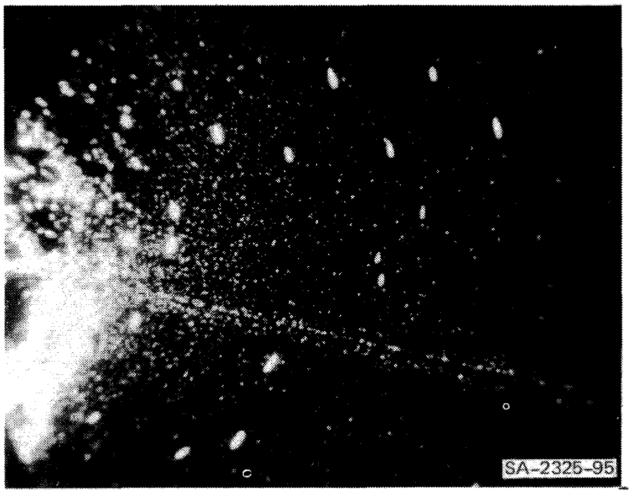


Figure 8—Effect of attenuating burst power by 20 dB

limit. Also shown is Shannon's bound for this power-limited gaussian channel. The spread-spectrum modulation scheme selected for the experimental repeater was chosen as a compromise between the relatively inefficient use of power achieved by the simplest scheme and the very complex receiver to achieve the most efficient scheme.

Also shown on Figure 9 is the multipath bound which occurs when the multipath delay-spread exceeds a single bit duration and introduces intersymbol interference. This will occur in worst cases at bit rates of 100 KB. Schemes to overcome this limitation include the use of spread-spectrum signals which change waveform at each bit, and the use of signals transmitting several bits per signalling element.

OTHER DESIGN CONSIDERATIONS

One problem of packet radio communications is generated by the need to share one common channel by many radio elements. Omni-directional radiation from radio transmitters increases the congestion in the channel. If a receiver is able to sort out or capture a packet from an ensemble of many packet signals impinging on its antenna, more users can be served by the network. The spread-spectrum modulation and detection techniques used in the experimental repeater are intended to provide a measure of discrimination so that one or more packets can be successfully detected in the presence of other packets. When there is no multipath, this discrimination can be achieved by sampling the detector output synchronously with one of the signals through a narrow sampling window. The probability of interference from an overlapping packet will be reduced by the ratio of sampling window width to bit duration. Since packets are transmitted in an asynchronous fashion, the receiver must synchronize itself to the transmissions to successfully detect a packet. Synchronization must be accurate enough so that peaks from successive bits fall within the sampling window. Furthermore, synchronization must be achieved rapidly to avoid long synchronizing headers which would add to the packet overhead. Multipath compounds the synchronization problem in several ways. First, it spreads out the received signal in time so that the probability of interference is increased and multiple-packet discrimination is decreased. Second, the multiple peaks (see e.g., Figure 3) are often of approximately equal amplitude so that small scintillations cause difficulty during sync acquisition and tracking. Finally, if the receiver (or transmitter) is moving a strong path may become weak and a weak path strong, so that the signal in the sampling window may exhibit deep fades.

A design consideration arising from the network structure of packet radio is the requirement that control and management of packet routing, flow control, and error detection is distributed throughout the network.[3] This requires that each station, repeater, and terminal in the network possess a minimal digital computation and control capability. Therefore, small, flexible, inexpensive processors with memories are needed in each element for the packet radio network to operate successfully.

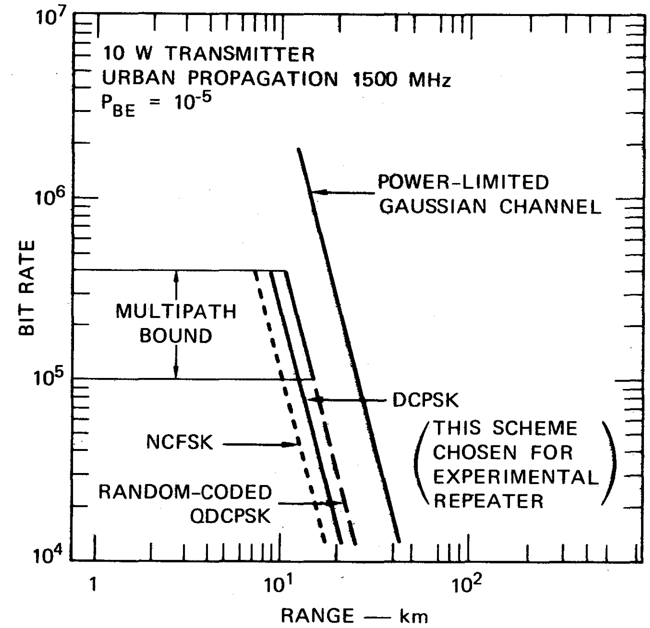


Figure 9—Comparison of some modulation schemes

Thus the rf channel and distributed network control requirements necessitate network elements with maximum rf power using complicated modulation, encoding, and synchronization techniques and with packet formatting, channel access, and network protocol software-controlled by a small CPU with memory. The implementation of such a component represents a challenge to current and future technology.

AN EXPERIMENTAL PACKET RADIO REPEATER

A repeater recently developed to support network experiments is a good example of the application of current technology to a packet radio network. The repeater element is chosen since it is an internal network element and invariant to the user interface or specific application. Additionally, it contains most of the needed functions of all network elements. The function of a packet radio repeater is to extend the range of terminal-station links, therefore, it must receive and retransmit packets. It must also invoke routing protocols and error control. For these reasons, the repeater must have access to the radio channel and contain logical decision capability to carry out the routing and error control tasks. The repeater consists of a radio, a digital processing section, and software for this processing. The description is divided into these three subelements.

The radio portion of the repeater is a transceiver that

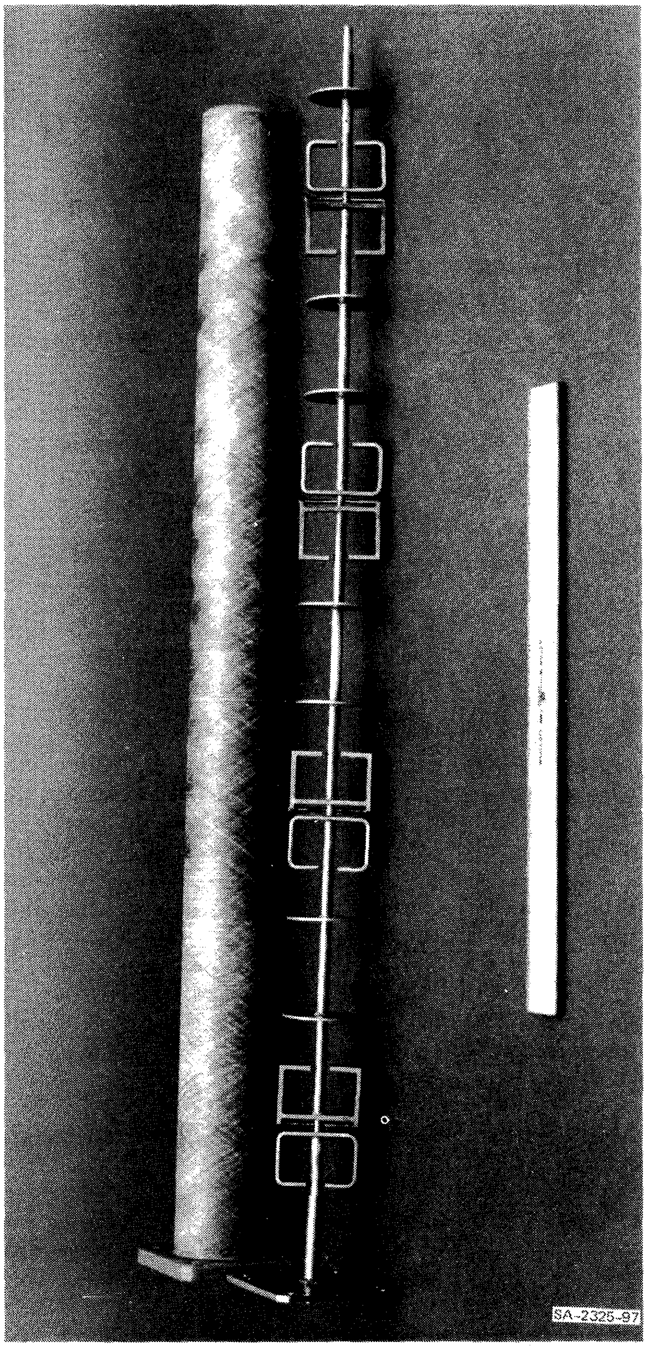


Figure 10—The experimental packet radio repeater antenna

operates in the 1 to 2 GHz frequency range. The transceiver transmits and receives over a common channel via one antenna and therefore operates in a simplex mode (receive and transmit modes are mutually exclusive). The repeater antenna shown in Figure 10 is a 4 element colinear array with omnidirectivity in the azimuthal plane and provides vertical directivity to obtain a gain of 8 dBu. The use of a single rf channel and omnidirectional antennas allows dynamic allocation of channel capacity over large areas using relatively simple network protocol.[3]

The signal processing section includes the encoding and modulation processes in the transmitter and the automatic gain control, demodulation, detection, and synchronization processes in the receiver. The encoding/modulation process functional diagram is shown in Figure 11. The data is differentially encoded to avoid the necessity of reconstructing a phase-coherent reference at the receiver. A read only memory (ROM) is used to store a pseudorandom spread spectrum chip code. The differentially encoded data gates the sequence or its complement to the impulse generator. The impulse generator impulses a 2-chip-long, cosine-weighted surface acoustic wave device (SAWD)[8] every chip interval. The resultant output of the SAWD is the transformation of a data bit into a multichip spread spectrum waveform. The chips are minimum phase shift keyed (MSK)[9] which yields a waveform that is constant amplitude and phase continuous.

The signal processes in the receiver are shown in the functional diagram of Figure 12. The received signal, after passing through the RF preamplifier, is down converted to the IF frequency. Then it is amplified by an automatic gain control (AGC) IF amplifier. The output of the AGC amplifier drives a SAWD matched filter which may be viewed as a tapped delay line. The SAWD property of nondispersive group-delay allows significantly long delays (up to many microseconds) in a 3- or 4-inch package. By proper placement and weighting of the taps along the device, any causal impulse response can be realized that is within the bandwidth of the device. The SAWD impulse response is reverse-time-matched to the pseudo-random spread spectrum code sequence described in the encoding/modulation process. When the received signal is identical to the SAWD impulse response, the output of the SAWD matched filter is the autocorrelation function of the sequence. A typical autocorrelation output response is

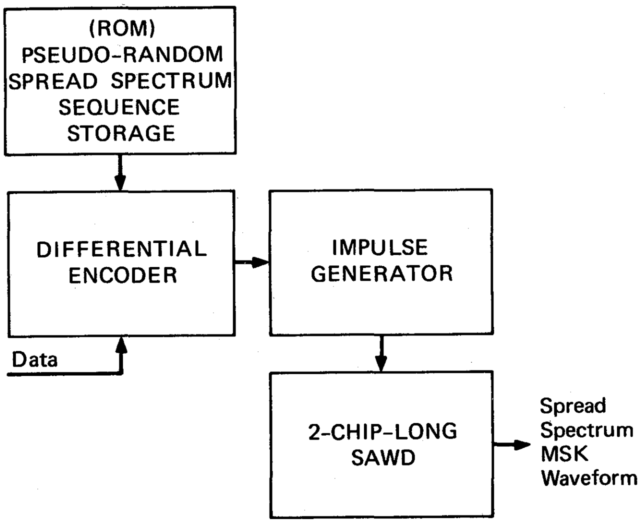


Figure 11—Functional diagram of encoder/modulator

shown in Figure 13. The signal passes through two identical SAWDs and the outputs of the two SAWDs are summed and subtracted in the 180° hybrid. This allows the decoding of differentially encoded data by comparing the pseudo-random sequence autocorrelation of a bit to the previous bit's autocorrelation. The sum and difference outputs are each AGC-amplified and envelope-detected on separate but identical channels. The outputs of the two channels provide the inputs to the data detector. The data detector consists of a differential comparator that compares the sum and difference inputs and decides the data is a 1 or a 0.

The other processes needed in the receiver signal processing are the bit synchronization circuit and the end-of-preamble detector. The bit synchronization circuit gates the data detector output through a narrow time window to provide time and multipath discrimination. The bit synchronization circuit phase locks to the sum of the difference and sum channels so that the time window is synchronous with the autocorrelation peaks of the SAWD matched filters. The purpose of the preamble detection circuit is to properly define the beginning of a packet. This is accomplished by attaching a preamble to the front of a packet which consists of a sequence of Barker codes.[10] The Barker codes were selected because of their high peak-to-sidelobe autocorrelation property. The preamble detector is a digital matched filter matched to the Barker bit sequence.

Assuming that repeaters are elevated above the ground, the path loss and multipath environment in repeater-repeater links will not be as severe as in the terminal-repeater links where the terminal is operating in a mobile environment. In order to take advantage of this and maximize network throughput, the repeater radio has two data detectors with different data rates. The lower data rate is 100 kB/s for terminal-repeater traffic and the higher data rate is 400 kB/s for repeater-repeater traffic. The receiver signal processing is, therefore, duplicated for these two data rates and the psuedo-random spread spectrum codes for the 100 kB/s and 400 kB/s are 128 and 32 chips long, respectively. This allows the occupied bandwidth to be held constant for both data rates. Careful selection of the pseudo-random codes for the two data rates provides low cross-correlation between the two rates and allows simultaneous reception of high and low data rate packets in the repeater radio.

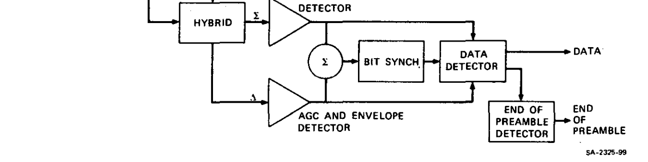


Figure 12—Signal processing circuits

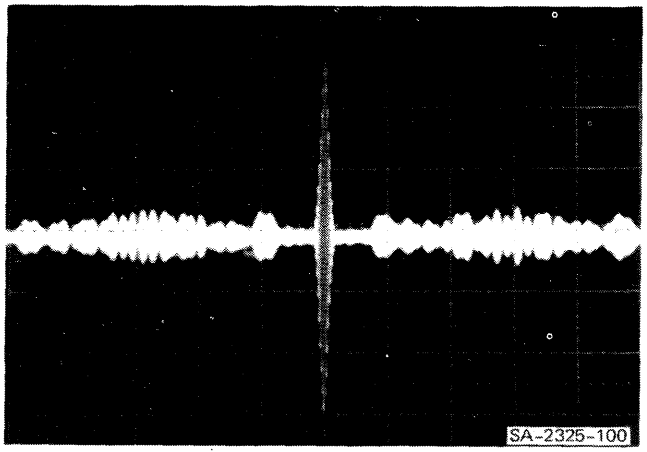


Figure 13—Typical autocorrelation response out of SAWD (128 chips/bit) Vertical scale: 10 mv/cm. Horizontal scale: 1 microsec/cm.

The functions of the RF head are frequency translations and power amplification. The application of thin film hybrid circuits to the repeater radio has represented an important technological tool. The integration of RF functions through the use of thin film hybrid techniques provides a means to reduce size and eliminate intra-connectors in the radio. Figure 14 (RF Head) shows an L-band RF head that contains an up converter, power amplification chain, transmit/receive switch, circulator, receiver preamplifier and filter all on thin film alumina substrates. This affords a significant savings in space, connectors and cabling. A functional block diagram of the RF head is shown in Figure 15.

The packet transport protocols[3] consisting of repeater initialization, packet routing, packet acknowledgments, and error control demand that the repeater possess significant logical processing power. The digital section of the experimental repeater provides this processing ability. It controls the radio and performs the following functions: Packet reception and transmission, error detection, packet routing protocols, and acknowledgment protocols. The hardware of the digital section consists of the CPU, address register decode, direct memory access control (DMA), radio interface and control, and memory.

The CPU is made up of National Semiconductor IMP-16 MOS devices. It has a 4 μsec machine cycle so that low power consumption C-MOS control logic can be used. The address decode provides for addressing up to 8K of 16 bit words. The DMA provides the interface between the radio transmitter and the two receiver detectors. All DMA ports are identical and provide direct access (write/read) to memory. The CPU initializes the DMA channels, which then proceed to access memory by stealing bus cycles with no further supervision from the CPU. The CPU is given priority on bus cycles. The radio interface and control performs bit serial to parallel conversions between radio and DMA channels and error detection using a cyclic code decoder. It also provides control of output power, center

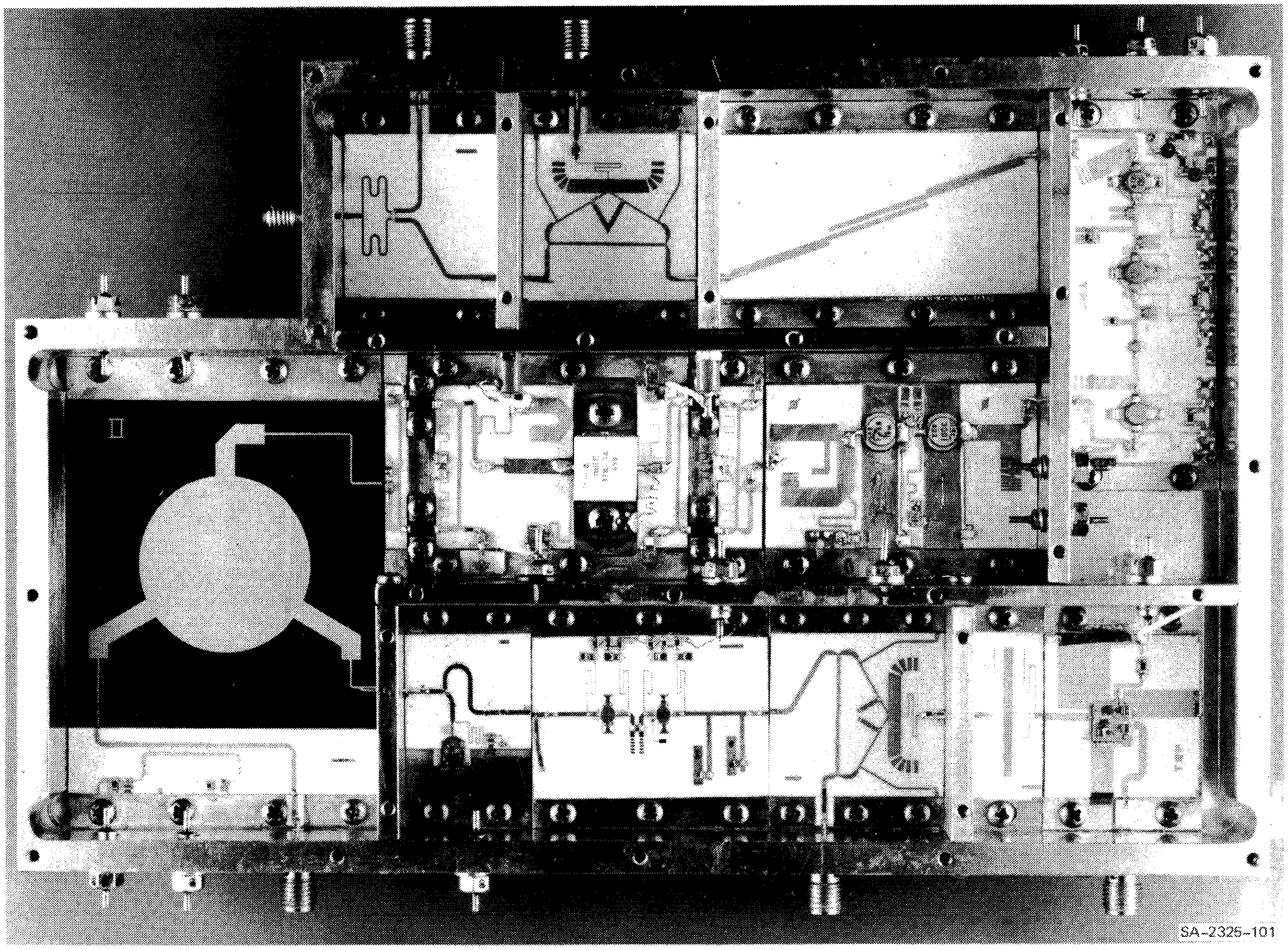


Figure 14—L-band RF head

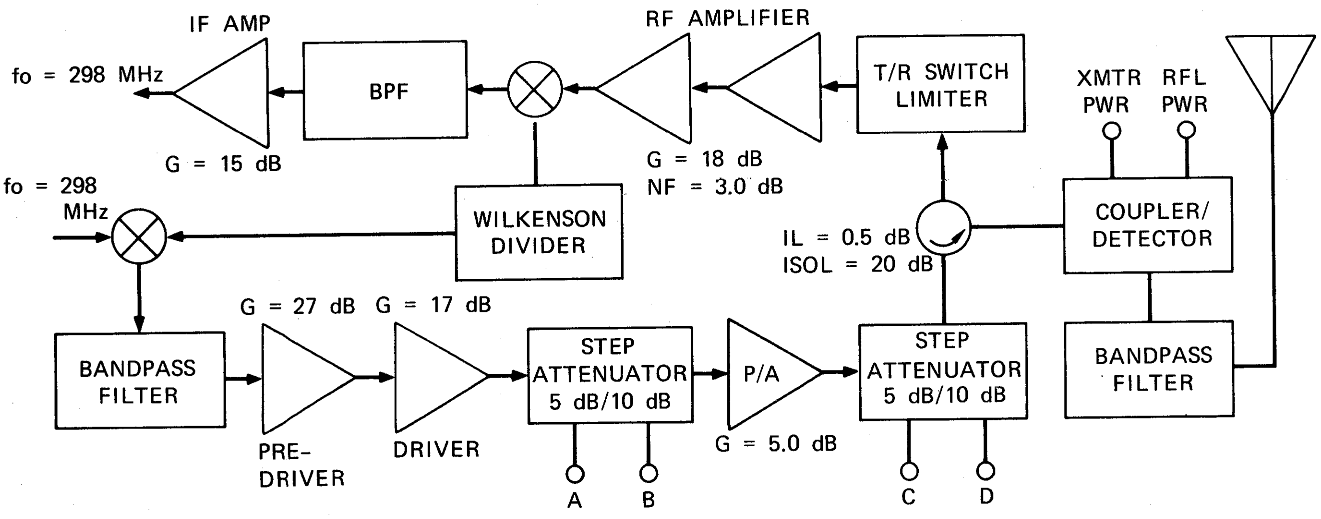


Figure 15—Functional block diagram of L-band RF head

frequency, transmit enable, and receive enable to radio. The memory consists of 3K words of random access (RAM) C-MOS memory for low power consumption and 1K words of programmable read only (PROM) MOS memory. The size and proportions of RAM to PROM are based on flexibility requirements for an experimental repeater.

The software for the experimental system is a multiprogrammed, interrupt driven system. Two independent programs coexist in the system and the state of the system is saved as control is transferred from one program to another. The operating system is interrupt driven with program control being transferred and processing initiated as a result of CPU interrupts. The system is structured into three programs which are defined as executive, background and foreground.

The executive program is utilized for operating system initialization, program control, and system test aids. The executive mode is noninterruptible. It transfers control between foreground and background while saving the state of programs. The executive removes power from selected repeater hardware elements when not in use and restores power when processing is reinitiated.

The foreground program contains the radio I/O packet handling processes. There are two levels of packet handling in the foreground program. The high level packet processing contains the routing and acknowledgment protocols. The low level packet processing contains the radio and DMA control and performs the actual handling processes under the direction of the high level packet processing. This modularization of packet handling provides flexibility for modifying packet transfer protocols without affecting the basic radio functions of transmitting and receiving packets.

The background program provides for overlay programs, on-line diagnostics, and performance monitoring. The overlay programs include programs that are received by the repeater and are executed and then overlayed. This feature allows for remote programming and reduces the memory requirements for resident software in the repeater.

The initial software in the experimental repeater resides in the 3K words of RAM and 1K words of PROM to provide flexibility. At the present, the executive program along with the debug control program, common routines, and the terminal I/O routines are resident in PROM. This leaves the foreground program and most of the background program in RAM. The packet buffers, diagnostic, and test programs are also in RAM. The projection of memory allocation for a packet radio repeater in an operational network would be 2K words of RAM and 2K words of PROM. Most of the software would reside in PROM with RAM memory devoted to packet buffers, temporary storage, base page, and overlay area.

Although the major goal of this first experimental repeater was to develop a versatile element to support network experiments, secondary goals to investigate and demonstrate applications of advanced technology to small light weight self-powered repeaters were also achieved. It was found that significant savings in size and power consumption could be achieved by applying thin film technology to integrate the RF functions in the RF head, and to integrate CMOS RAM chips in the memory. It was also found that power consumption could be reduced by powering down the CPU, modulator, and power amplifier chain when not in use. Using these techniques, a repeater has been constructed which is one cubic foot, weighs 40 pounds, and requires 25 watts average power.

In concluding the discussion of the experimental packet radio repeater, it is appropriate to identify what features of the repeater are most likely to change or be improved. It is most certain that the physical size will be reduced by using LSI techniques in the digital section and hybrid thin film circuitry in the radio section. Another area where improvement is probable is in spread spectrum processing gain. The present repeater has a maximum spread of 128 (21 dB processing gain). Spreads in the thousands can be obtained with convolver surface wave devices. The ability to change spread spectrum codes on a packet or bit basis is another feature which will further enhance the repeater's antijam and antispoof capability. Other features to be forthcoming are full mobile radio operation via diversity techniques and higher data rates, made possible using m-ary coding or convolvers with multipath suppression capability. A more detailed discussion of the application of future technology to the development of packet radio network elements follows and addresses several of these issues.

APPLICATIONS OF FUTURE TECHNOLOGY

Although a packet radio network using small microprocessor-based repeaters is feasible with today's technology, a great deal of work remains to be undertaken before such a network is a realistic practical alternative.

In this initial effort to develop an experimental repeater, every attempt has been made to retain versatility and flexibility so that the repeater will support a wide variety of experiments. New technological developments will be incorporated as they become available and as more is learned about packet radio networks. Future technological advances which seem particularly promising include advances in SAW technology to achieve a greater variety of devices, and applications of semiconductor technology to achieve LSI and VLSI memories, CPU's and interface circuits.

Surface Acoustic Wave devices can be applied to a large variety of rf signal processing problems.[8] Simple bandpass filters using SAW technology would integrate easily into rf thin-film circuits and might replace bulky coaxial filters to reduce size and weight. SAWDs may also find application in the feedback structure of oscillators.

Size, weight, and power consumption can also be reduced by the application of LSI and VLSI to both analog and digital circuits currently built with discrete components and MSI chips. The most likely areas of improvement here are in the modem, the I/O interfaces, and the microprocessor. The microprocessor memory size may be reduced when a single network protocol is established as the best. When this occurs, the software can be microcoded to reduce ROM size. New ROM chips with more bits of storage, using less power are also probable as semiconductor technology advances. As the network and component design firm up interfaces will be standardized so that it will be worthwhile to develop single chip LSI interface circuits.[11]

It will be desirable to obtain a faster microprocessor since this will provide higher throughput and less time delay through the network. Higher speed, lower power processors soon to be available[12] seem to offer an order of magnitude improvement in both speed and power consumption. According to all indications, microcomputer technology will change at a tremendous rate. The development of Silicon on Saphire (SOS), integrated injection logic (IIL) and low power Schottky (LPS) devices all point toward the day when as few as four chips may replace the 13 cards presently in the digital section of the repeater. Ideally, the four chips will be: a CPU chip with 500 ns cycle time, a 2K×16 bit 100 ns RAM, a 4K×16 bit 100 ns ROM, and a multiport byte-parallel and bit-serial interface. The total power consumption should be less than 1 W.

The experimental repeater was designed to test new modulation technologies to operate in difficult rf environments. Two directions of change are possible here. Much simpler modulation techniques can be applied to packet radio nets to operate in less hostile environments, and more complex techniques can be applied to improve performance in the most hostile environments. The ALOHA net at the University of Hawaii is a good example of the former possibility. In this case, the straightforward application of thin-film hybrid technology to the RF portions, and LSI technology to the lower frequency portions of the radio and modem will reduce size and power by an order of magnitude.

We expect that new SAW devices such as convolvers will improve operation of the system in hostile environments. A convolver is a device which accumulates the cross-product of two signals.[13] Such a device has an advantage over the usual fixed-impulse response SAWD, since any causal impulse response within the device time-bandwidth constraints can be obtained, and the impulse response can be varied with time. Using a convolver it would be possible to eliminate most intersymbol interference caused by multipath by changing the transmitted waveform (and receiver matched-filter) for each bit. If waveforms are chosen to be pseudo-orthogonal, then one symbol will not interfere with the next. The use of a convolver can, however, make synchronization more difficult since the impulse response depends upon a local reference and the reference must be in rough synchronization with the received waveform. Nevertheless, a convolver may allow higher bit rates over short ranges.

A major requirement in all future designs will be to reduce average power consumption so that the power-source size and weight can be reduced. Any improvements in portable power source energy-density will result in a much more flexible network. Although much effort is being expended to develop such improved sources, none seem to be on the immediate horizon. Perhaps this is the one area of technology in which there is a major gap between need and availability.

CONCLUSION

We have focused on the impact of rf channel, network design, and technology on the repeater because it is the critical communications element in a packet radio network. It can be converted to a station by adding a minicomputer, and to a terminal by adding appropriate I/O components. Since a repeater has been successfully implemented, it is reasonable to conclude that a packet radio network is technically feasible. Future experiments must be aimed at demonstrating operation of an entire network.

ACKNOWLEDGMENT

The authors wish to thank the members of the Advanced Research Project Agency research team designated as the Packet Radio Working Group who have contributed ideas, concepts and specific design details to the work discussed in this paper.

REFERENCES

1. Kahn, R. E., "The Organization of Computer Resources into a Packet Radio Network," *National Computer Conference* 1975, Anaheim, California, Proceedings of this conference.
2. Abramson, N., "The ALOHA System—Another Alternative for Computer Communications," *Proceedings of AFIPS 1970 Fall Joint Computer Conference*, Vol. 37, pp. 281-285.
3. Frank, H., I. Gitman and R. Van Slyke, "Packet Radio System: Network Considerations," *National Computer Conference* 1975, Anaheim, California, Proceedings of this conference.
4. Fralick, S., D. Brandin, F. Kuo and C. Harrison, "Digital Terminals for Packet Broadcasting," *National Computer Conference* 1975, Anaheim, California, Proceedings of this conference.
5. Burchfiel, J., R. Tomlinson and M. Beeler, "Functions and Structure of a Packet Radio Station," *National Computer Conference* 1975, Anaheim, California, Proceedings of this conference.
6. Okumura, Y., E. Ohmori, T. Kawano and K. Fukuda, "Field Strength and Its Variability in VHF and UHF Land-Mobile Radio Service," *Rev. Elec. Commun. Lab.*, Vol. 16, pp. 825-873, September-October 1968.
7. Nielson, D. L., *Microwave Propagation and Noise Measurements for Mobile Digital Radio Application*, SRI Packet Radio Note No. 4, January 1975, Stanford Research Institute, Menlo Park, California.
8. Holland, M. G. and L. T. Clairborne, "Practical Surface Acoustic

Wave Devices," *Proceedings of the IEEE*, Vol. 62, pp. 582-611, May 1975.

9. Doelz, M. L. and E. T. Heald, *Minimum-Shift Data Communication System*, U.S. Patent No. 2,977,417, March 28, 1961.
10. Barker, R. H., "Group Synchronizing of Binary Digital Systems," *Communication Theory*, W. Jackson, Ed. 1953, pp. 273-287.
11. Young, L., T. Bennett and J. Larell "N-Channel MOS Technology Yields New Generation of Microprocessors," *Electronics*, pp. 88-95, April 1974.
12. Schmid, Herman, "Monolithic Processors," *Computer Design*; Vol. 13, No. 10, October 1974.
13. Smith, J. M. and E. Stern, "Surface Acoustoelectric Convolvers," in *Proc. 1973 IEEE Ultrasonics Symp.*, Monterey, California, November 1973, pp. 142-144, IEEE Order 73CHO807-8SU.

Functions and structure of a packet radio station*

by J. BURCHFIEL, R. TOMLINSON and M. BEELER
Bolt Beranek and Newman
Cambridge, Massachusetts

INTRODUCTION

A packet radio network is a digital broadcast channel, fixed and mobile digital terminals which are sources and sinks of information, stations which provide centralized routing control and interconnections to other networks, and repeaters which provide area coverage for mobile terminals by performing a store-and-forward function on the radio broadcast channel. An experimental testbed for these concepts is now under construction at the Stanford Research Institute, Palo Alto, California to provide experimental validation of theoretical models and simulation predictions.[1-6]

Objectives of this technology exploration include low-cost use of a broadcast band in digital burst mode to support digital computer and terminal communication, demonstration of coexistence with existing broadcast applications, and secure mobile communications through encryption techniques.

The Packet Radio Network (PRN) concept was initially explored in the ALOHA Project at the University of Hawaii.[2] Such a network is different from point-to-point packet switched networks in a number of significant areas:

1. The use of mobile terminals requires facilities for tracking and handoff of the terminals from repeater to repeater and station to station.
2. The use of a common broadcast channel results in collisions due to simultaneous packet transmissions: the communications paths require extensive error checking and correction to handle the resulting high error rate.
3. The repeater is required to operate unattended in relatively remote areas for long periods of time. It must therefore be a simple, low-powered component. Accordingly, it is not designed to support complex dynamic routing algorithms. Instead, such functions must be provided in the stations, giving some measure of centralized control in the Packet Radio Network.

On the other hand, a PRN has many features and functions in common with a point-to-point packet switched network:

1. The PRN must provide routing which adapts dynamically to component failures.
2. The PRN must support interprocess connections which have flow control and error control.
3. The PRN must have centralized monitoring, debugging, and statistics collection tools to provide maintenance and performance evaluation capabilities.

The dynamic packet routing capability (packet store-and-forward) is programmed in the repeaters, with the stations providing initialization and centralized control of parameters for terminal tracking. The programmable capability of the repeater is provided in its IMP-16 microprocessor.

Reliable data transmission between PRN data sources and sinks is required in spite of errors and transmission 'collisions' on the broadcast channel. This is achieved by defining a logical entity called a 'connection' between the source process and destination process, and performing end-to-end error detection and correction over this noisy channel. The connection is thus a reliable (error corrected) data transmission path.

The programs for providing interprocess connections within the PRN must be programmed into each terminal (data connection), each repeater (control connection), and each station (data and control connections).

In addition to the communications support, the terminal must also have a terminal handler program which manages terminal input and output buffers and performs translation of format effector characters as needed.

The programs which provide centralized monitoring, debugging, and statistics collection are located in the station, with a small (slave) routine in each repeater. These functions are shown in Figure 1.

The set of functions which appear in common in a station, repeater, and terminal are identified in Figure 1 as a Packet Radio Unit, or PRU, which has been implemented as a standard piece of hardware and software by Collins Radio. It will serve standalone as a repeater; addition of the station interface hardware and software option converts it to a station; addition of the terminal interface

* This research was supported in part by the Advanced Research Projects Agency of the Department of Defense under Contract No. DAHC15-71-C-0088.

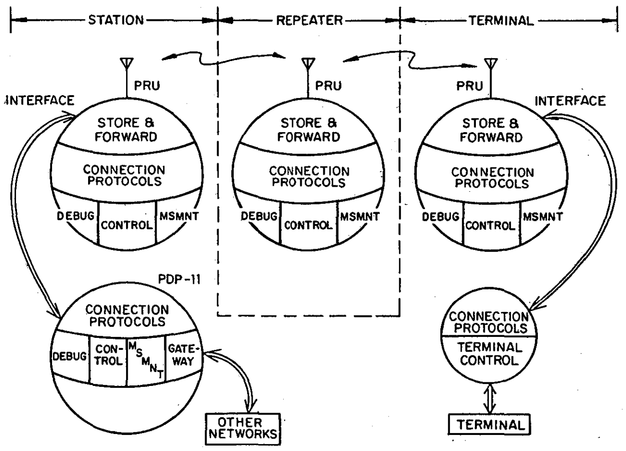


Figure 1—Functions of a packet radio station, repeater and terminal

hardware and software option converts it to a terminal. The additional functions shown for terminal may either be implemented in a separate microprocessor or provided in a separate memory partition within the terminal's PRU, timesharing its microprocessor for economy.

Our prototype station has the additional station functions implemented in a Digital Equipment Corp. PDP-11, which is also interfaced to the ARPANET.

An implementation of the station described in this paper will provide control functions for an experimental test of packet radio concepts in a testbed packet radio network in the Palo Alto, California area during 1975.

CONNECTIONS IN THE PACKET RADIO NETWORK

One of the basic facilities required in the PRN is support of interprocess connections which provide reliable delivery of data from a PRN source to a PRN destination. Such connections require flow control to prevent a source from overloading the network and causing serious congestion. They also require error control because of message interference in the shared broadcast communication channel. The error control mechanisms selected are:

1. A sequence number in each packet to permit detection of missing or duplicate packets.
2. An end-to-end positive acknowledgment for packets which arrive successfully.
3. A source timeout which causes periodic retransmissions of unacknowledged packets.

Since most expected uses of the PRN will require bi-directional communication, the PRN connection is bi-directional, with flow control and error control information for data transmission in each direction piggybacked onto the data flow in the opposite direction. This arrangement is depicted in Figure 2.

Certain applications of packet radio will not require reliable delivery of data (e.g., real time seismic or speech data), and such devices will not be required to support the connection protocol described above.

Figure 2 shows that multiple packets of data can be in transit in both directions at one time. The acknowledgments are cumulative, i.e., an acknowledgment carrying serial number 9 acknowledges successful receipt of all packets up to and including packet number 9. It is therefore unnecessary to send out an acknowledgment for each data packet received; one 'ack' can cover a number of data packets. Further, the 'ack' carries such a small amount of information, that it can be inserted into the header of a data packet travelling in the same direction, resulting in very low overhead for this acknowledgment mechanism. However, timely acknowledgment is required to prevent source timeout and retransmission, so if there is no data traffic which can be used in this piggyback fashion for 100 milliseconds or so, a separate 'ack' packet must be generated and sent.

The status information which must be maintained at each end of the connection is minimal: four sequence numbers. The values of the sequence numbers at the two ends differ by the traffic currently 'in the pipeline'. Flow control is established by the convention that the sender can only send up to N packets ahead of the last packet which was acknowledged. Equivalently, a source may not have more than N packets 'in the pipeline' at one time. To keep the repeater code as simple as possible, N should be equal to one (packet-at-a-time) for repeater control connections. This protocol is a simplified subset of the protocol developed by Cerf and Kahn[7] for internetwork communications.

The protocols of the Packet Radio Network are layered, or hierarchical. The program which deals with control information at level M passes control and data for all levels greater than M as transparent data. Conversely, the program which deals with control at level M does not see control information at levels less than M; it is inserted by lower level programs on transmission, and stripped off by lower level programs on reception.

Figure 3 shows this layering explicitly: the connection protocol described above is the level 2 protocol, based on the level 1 routing protocol which controls the PRN store-and-forward routing for the packet. The routing protocol is itself based on a level 0 'Radio Hop' protocol which provides broadcast synchronization and error detection for transmission of the packet from one PRU to the next.

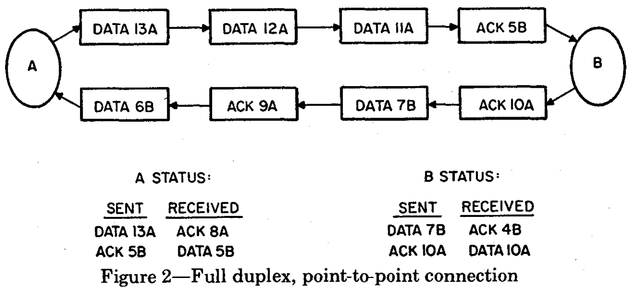


Figure 2—Full duplex, point-to-point connection

The sequence numbers shown were discussed previously. The 'sequence control' field is used to synchronize sequence numbers when a connection is established, and to signal termination of the connection. The 'function fields' provides an address: within a PRU, it selects the control process, the debugging process, or the measurement process.

STATION CONTROL FUNCTIONS

The control functions performed by a station include initialization of the PRN, dynamic routing changes, and multi-station coordination. Initialization of the PRN includes the following steps:

1. Measurement of RF propagation connectivity between all stations and repeaters. This measurement data is used to construct the *connectivity matrix*: this is a matrix of binary values which indicate the radio units which are capable of direct communication with each other.
2. Configuring the PRN by loading each repeater with routing parameters which control the packet store-and-forward program. These parameters specify forwarding of packets [in the direction of minimum distance] to the next repeater within "earshot".
3. Establishing control, debugging, and measurement connections from the station to each repeater that it controls. These connections remain open to perform the indicated functions as long as the station and repeater continue to function normally.

The initialization algorithm must be able to bring up the network from a cold start, with no information in any station. It must permit any element to enter the net 'at any time on its own initiative, e.g., when a terminal is powered up the algorithm must connect it into the net. This algorithm should also provide a continuous monitor on connectedness to identify component failures. Finally, it should be quick, accurate and use minimum traffic.

The proposed procedure to meet these requirements is as follows:

1. Repeaters have two states: labelled (containing routing parameters) and unlabelled.
2. Each PRU (station, repeater, terminal) has a unique, hardwired I.D.
3. All repeaters send "search" packets at random times with some average rate dependent on whether or not the repeater is labelled.
4. A search packet states whether or not the repeater is labelled, gives its repeater I.D., and the I.D. of the station which labelled it.
5. All labelled repeaters that hear a search packet on the first hop forward it via their established route to every station that has labelled the receiving repeater. All unlabelled repeaters ignore search packets.
6. The station uses search packets arriving to generate the connectivity matrix, and relabel (or initially label) the net.
7. On initial entry by a station into an unlabelled net:
   a. The station listens for at least one search packet.
   b. If it hears none it should try to stimulate receivers in earshot.
   c. The first search packet heard is labelled as soon as heard.
   d. The next search packet may come through the labelled PRU or directly or both. The station labels the new PRU and notes whether the first PRU could hear it, and starts the connectivity matrix.
   e. The station continues to label or relabel as repeaters report in.
8. If an ability to trigger search packets is available, the process can be speeded up. This should allow stations entering a labelled net (one where another station is operating) to do so more rapidly. Repeaters should also have ability to trigger local repeaters, or to get local repeaters to bounce back search packets. This will occur in the above process for all labelled local repeaters, but requires special consideration for unlabelled PRU's.

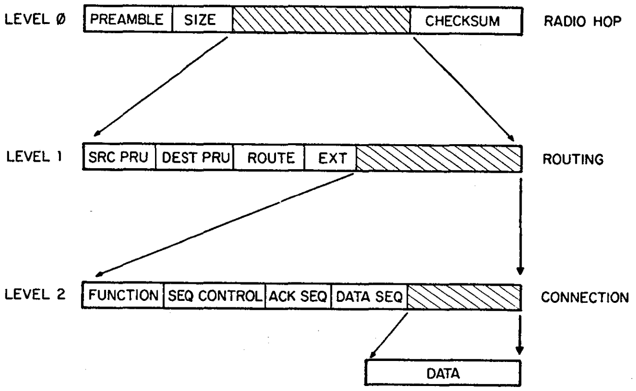


Figure 3—Protocols for support of a PRN connection

Once the station has labelled all PRU's and established connections to them, the information for maintaining these connections is entered into the station's *connection table.* This contains the status information described above for handling the connection protocol. As terminals come "on-line" within the PRN, each terminal is also given a connection to its controlling station, and this information is added to the station's connection table.

Dynamic routing changes are performed locally within the PRN by permitting a repeater to specify an alternate address for the next hop after some number of unsuccessful attempts to forward the packet along its specified route. This capability provides a localized terminal tracking facility which will hand off a mobile terminal from one repeater to another.

A less dynamic but more global routing change is required when a repeater or station fails: persistent alternate routing of packets signals this failure, and the station detecting it must reconfigure the PRN to route all traffic around the failed element. The station performs this reconfiguration by updating the connectivity matrix to indicate that the failed element is incommunicado; by recomputing minimum distance routes to the elements which are still active; and finally, by updating repeater routing parameters to route packets along these new routes.

The third station control function is multi-station coordination. This will be supported by PRN connections between stations and a station-station protocol. In some cases, when both stations are connected to a point-to-point network, it may be more efficient to use this network for station coordination rather than the PRN. This coordination is required in two cases:

1. Hand off of a mobile terminal as it passes out of the area controlled by one station and into the area controlled by another station.
2. An alternate station assuming all communications responsibilities for a station which fails.

These applications will require a distributed data base stored redundantly at a number of stations. The data will give status information on each active connection, and will be updated often enough that little information will be lost in the event of a station failure. Some of the PRN terminals may be accessing host resources in the ARPANET: an ARPANET reconnection protocol will be defined to permit switching the ARPANET portion of the terminal connection from the failed station to the backup station. These complexities are not being addressed in the initial experimental implementation, but they will be vital reliability measures in an operational packet radio network. Many of these issues of reliability achieved through maintenance of redundant distributed data bases have been explored in the RSEXEC (Resource Sharing Executive) distributed computation testbed.[10]

A second phase of the experimental program being conducted at the Packet Radio testbed network at Palo Alto will demonstrate the concepts of redirecting traffic from a failed station to an operational station, and the backup of resource allocation information.

STATION DEBUGGING AND STATISTICS FUNCTIONS

A level-3 debugging protocol has been defined which supports debugging of remote PRU's from terminals attached to a central station. The debugging functions include examining and depositing words into the PRU memory, and setting "mousetraps" which send an error code to the controlling station when some anomalous condition occurs, e.g., hardware failure. These functions make it possible to do centralized software maintenance of remote, unattended repeaters. The maintenance terminal of the station will normally be attended by an operator or system programmer.

A similar mechanism permits centralized collection of traffic statistics, both through examination of counters in PRU memory and through centralized reception of special status conditions such as "trace packets" moving through the network. Again, it was essential to centralize this function for remote, unattended repeaters.

Once the station has collected a set of traffic statistics, it will normally forward these measurements to a service host for detailed statistical analysis, logging and plotting.

STATION SUPPORT OF NONENCRYPTED TERMINALS

Some PRN applications will require secure or private communications, i.e., end-to-end encryption between a terminal and the service host which it is accessing. Assuming that the station is merely an intermediate node in the path from terminal to host, the station must be completely transparent to the scrambled data. (Any modification of the data could make it impossible to unscramble.)

On the other hand, some PRN applications do not require encrypted terminals: for these applications, the station can take over part of the terminal service function and simplify the terminals. One example of this is the TELNET RCTE option. [Reference 9] TELNET is the protocol which translates a variety of different physical terminals into a single standard logical terminal called a network virtual terminal: this conversion may involve both code conversion and interpretation of format effector characters (tab, carriage return, etc.). This concept provides and extremely valuable simplification because previously N*M conversion routines were required to interface N terminals to M systems. Using the network virtual terminal concept, only N+M conversion routines are required.

The RCTE option of TELNET is Remote Controlled Transmission and Echoing. It permits character echoing for a full duplex connection to be performed local to the terminal (eliminating annoying echoing delays) under control of the remote service host, which may, for example, suppress echoing of typed-in passwords.

The TELNET RCTE protocol option may be too complex to incorporate into the simplest unencrypted PRN terminal. In this case, the station can handle the protocol conversation with the remote server on the terminal's behalf. Of course, the station cannot perform this service for encrypted terminals.

The TELNET process in the station may also be used to specify and set up connections to remote server hosts via a gateway connection.

INTERNETWORKING APPLICATIONS

So far, we have focussed on the attachment of terminals to a PRN. It is also reasonable to attach service host computers to a PRN, particularly when the network may be deployed in the absence of other communications networks, (e.g., for fleet communications). Such a host attaches to the PRN as a multiplexed set of the standard PRN connections described earlier.

When some other network is present, it is important to provide connections between the terminals and hosts of the PRN and the terminals and hosts of the other network. This is being done for the ARPANET in two ways:

1. For communication with ARPANET hosts which support a protocol congruent with the PRN connection protocol (the Cerf-Kahn protocol mentioned previously qualifies here), the station functions as an extremely simple gateway: arriving packets are simply forwarded into the other network after their header format is converted to that of the destination network. In this case, the station does not detect missing or duplicate packets, and does not reorder packets which arrive out of order; it is merely a packet reformatting and readdressing service.
2. The second approach will be conversion between the host-host protocols of the two networks. In particular, one connection will be established from a PRN device to a station using the PRN connection protocol described in an earlier section. Another full-duplex connection will be established from the station to an ARPANET host using the current ARPANET host-host protocol. Data arriving from either of these connections will be forwarded through the other connection.

The first approach has four obvious advantages beyond its simplicity: first, error control is truly end-to-end instead of two path (PRN device-to-station, station-to-ARPANET device). Failure of the station will not cause data loss with this approach though it would with the two-path approach.

The second advantage of this approach is dynamic rerouting after a station failure. Since the station is merely doing packet readdressing, any other station attached to the PRN and the ARPANET can do this function equally well; no connection status information is kept in the station. Recovery and continued operation of a connection after the station fails merely requires redirecting the packet traffic to another gateway station.

The third advantage of the "simple gateway" approach is that the gateway does not introduce additional delay into the end-to-end path by forcing reassembly and reordering of packets at an intermediate location (the station).

The fourth advantage of the "simple gateway" approach is that it supports end-to-end encryption of all data except the address headers. This can provide security against data disclosure, but no security against traffic pattern and volume analysis. The second gateway approach would require a secure station to decrypt and re-encrypt data flowing through it.

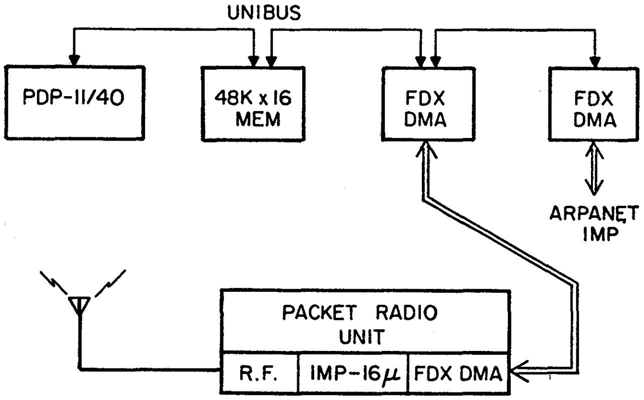


Figure 4—Station hardware

On the other hand, the second approach enjoys two advantages: first, there is independent flow control on the two connections, so 'acks' for PRN packets are returned from the station (about 200 msec) rather than from the remote host (about 1 sec). This faster turn-around of acknowledgments permits shorter source timeouts and more rapid recovery from lost packets.

The second advantage of this approach is that it permits the utilization of completely different protocols which are currently operational within the two networks, as long as the two protocols are functionally equivalent and a mapping exists between the two. This applies not only to host-host protocol, but also to higher level protocols such as TELNET, file transfer, and remote job entry. All that is required is a conversion program for the high-level protocol of interest, interposed between the PRN connection and the ARPANET connection.

The experiments planned for the packet radio testbed system will explore the tradeoffs and performance advantages of these two different gateway concepts in detail.

STATION STRUCTURE

Figure 4 shows the hardware organization of our prototype station: it is a PDP-11 processor interfaced to a packet radio unit. In the initial tests, it will also be connected as a gateway to the ARPANET. The hardware interface between the PDP-11 and the PRU consists of a pair of memory channels on each end, permitting full duplex DMA packet transfers.

The software organization inside the PDP-11 is shown in Figure 5. We selected the ELF operating system* as the basic environment. This is a time-sharing operating system written in PDP-11 assembly language. Two modifications were required to ELF to support the packet radio application: first, the PRU was added as a bidirectional ELF device. This required addition of an interrupt service routine for the PRU interface hardware, and send and

* Developed by the Speech Communication Research Lab., Santa Barbara, California.

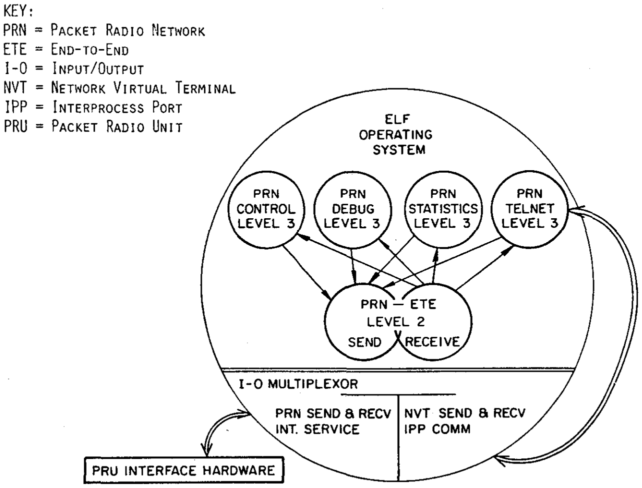


Figure 5—Station software

receive subroutines within the ELF I-O multiplexor module. These routines were coded in assembly language and integrated into the ELF operating system. The second change required to ELF was the 'Network Virtual Terminal' support package which permits PRN terminals to appear identical to local ELF terminals (attached to the PDP-11) by use of a connection to the PRN terminal, a protocol-handling PRN TELNET process, and ELF's interprocess communication facility. These routines were also coded in PDP-11 assembly language and integrated into ELF. The use of internetwork protocol[7] will permit PRN terminals to access services of hosts on other networks.

It was possible to do the remaining packet radio software as independent user applications processes which execute under ELF. Accordingly, they were programmed in BCPL (Basic Compatible Programming Language) to obtain all the clarity, debugging, maintenance, and extensibility advantages of high-level programming. These modules are not part of ELF, and they are not integrated into the operating system.

First, a pair of processes was created which send and receive packets to and from the PRU device previously integrated into ELF. The send process and receive process share a common address space which contains the connection table: this holds all status information for every connection being maintained by the station. These two processes are responsible for handling the PRN connection protocol described previously.

Within the same program module is a subroutine for creating a connection. This subroutine takes the address of the destination PRN device, consults the radio propagation connectivity matrix, and constructs the route which should prefix all packets sent to the specified destination. This route is stored as part of the connection status in the connection table when the connection is established.

The control process is another independent process for initializing the network and causing dynamic routing changes in response to changes in repeater-terminal propagation connectivity or repeater failures. It is responsible for keeping the propagation connectivity matrix up-to-date. This matrix is shared with the connection initialization routines which find a route to the requested device.

In the PRN initialization procedure, the control process calls on the connection module to establish a control connection to the station's PRU. It sends commands over this connection to trigger connectivity measurements (exploratory packets which request answerback from stations and repeaters within earshot). As measurement information comes back on this connection, the control process fills in entries in the connectivity matrix, and establishes control connections to the PRU's of the newly-discovered devices. This procedure is iterated until every station and repeater in the area has been configured into the network. At this point, the control process has an open control connection to every other station and repeater in the PRN.

When any repeater detects a significant routing event, e.g., failure of some previously established route or a request from a terminal to enter the network, the repeater forwards this information over its control connection to the nearest station. When a terminal comes on-line, the station establishes a data connection to it and provides an 'information service' to assist in completing the connection to the destination device, which may be either in the PRN or in the ARPANET. When repeater connectivity changes the station will update the connectivity matrix and reconfigure the network to bypass the failed link by modifying the routing control parameters in the effected repeaters.

The debug program is another independent process. On request from the maintenance terminal, it calls on the connection module to open a debugging connection to the PRU of interest. The debugger sends commands over this connection to examine or deposit words in the PRU's microprocessor memory, and the PRU responds with a positive acknowledgment for each command. There are also commands for setting traps on anomalous program conditions. When one of these conditions is encountered (assuming the PRU is still operational) it sends the appropriate trap code over the debugging connection to the debugger. This types out on the controlling terminal, which is presumably attended by a system programmer or operator.

The statistics collection module is another independent process which gathers data both by examining PRU memory, and by receiving statistics trap conditions spontaneously emitted by PRU's. This operation parallels the operation of the debugger described above.

Finally, the PRN TELNET process performs the second type of gateway function described above: conversion between the PRN connection protocol and the ARPANET host-host protocol. Terminals on the PRN appear identical to the terminals attached to the PDP-11, and are able to access remote ARPANET service hosts in the same way.

CONCLUSION

Successful demonstration of the packet radio concepts will lead to new digital communications services for mobile terminals which are reliable, difficult to detect or jam, and which make efficient use of the electromagnetic spectrum by providing coexistence with current uses of the spectrum. Possible applications include both command and control communications and secure digital voice.

The system structure described above will provide reliable, efficient and maintainable support for packet radio network communications.

REFERENCES

1. Kahn, R. E., *The Organization of Computer Resources into a Packet Radio Network,* these proceedings.
2. Abramson, N., R. Binder, F. Kuo and W. Okinaka, *ALOHA Packet* "Broadcasting, A Retrospect," these proceedings.
3. Garrett and S. Fralick, "A Technology for Packet Radio," these proceedings.
4. Kleinrock, L. and F. Tobagi, "Random Access Techniques for Packet Radio Networks," these proceedings.
5. Frank, H., R. Van Slyke and I. Gitman, "Packet Radio Network Design, System Considerations," these proceedings.
6. Fralick, S., D. Brandin, F. Kuo and Harrison, "Digital Portable Terminals," these proceedings.
7. Cerf, V. and R. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Trans. Comm., May 1974.*
8. Crocker, S., J. Heafner, R. Metcalfe and J. Postel, "Function Oriented Protocols for the ARPA Computer Network," *AFIPS Conference Proceedings,* Vol. 40.
9. Crocker, D. and J. Postel, *Remote Controlled Transmission and Echoing Telenet option.* RFC #581, Network Information Center.
10. Thomas, R. H., "A Resource Sharing Executive for the ARPANET," *AFIPS Conference Proceedings,* Vol. 42.

Digital terminals for packet broadcasting*

by STANLEY C. FRALICK and DAVID H. BRANDIN
Stanford Research Institute
Stanford, California

and

FRANKLIN F. KUO and CHRISTOPHER HARRISON
University of Hawaii
Honolulu, Hawaii

INTRODUCTION

Roberts illustrated the potential use of packet switching technology by postulating a personal computer terminal using radio broadcasting to connect the user to a computer.[1] The proposed terminal had a unique five-finger keyboard and plasma-discharge display. The keyboard would generate and send characters, one at a time, to the computer using 64 bit packets per character. The computer could convert these to a 35-bit (5×7) pattern and retransmit a 144-bit packet to the terminal to control a 5× 7 dot matrix character. Thus, the terminal needed no character generation logic and only a minimum of digital control logic to interface keyboard and display to a radio modem. This was a reasonable concept insofar as the terminal was intended to operate within a short distance of the computer to accommodate low-power radios, and so long as only a few terminals were in use.

Roberts assumed a random access packet broadcasting transmission mode formerly developed by Abramson[2] and now known as the pure ALOHA technique. Under a pure ALOHA mode of operation, packets are sent by the terminals to the central station computer(s) in an unsynchronized manner. In this scheme the lack of positive acknowledgments (POSACK) controls retransmissions, as necessary. Using the pure ALOHA technique with a 10-character per second terminal and assuming 64-bits per character (a peak data rate of 640 bits per second), it can be shown that a 100 kilobits per second channel will simultaneously support only 26 terminals.[2] To accommodate more terminals, higher bit-rate channels are needed, along with a more efficient packet structure. For example, with Robert's proposed packet structure, modified by sending 10 characters instead of one character per packet, the same channel will support twice as many terminals.

Higher bit-rates require more transmitter power for the same range. Greater efficiency requires more memory and logic in the terminal. It has been found that the size, weight, and power consumption of the radio transmitter will dominate the terminal at high bit-rates unless the terminal range is small. To obtain larger coverage areas, a network of radio repeaters is needed. Because of the random nature of propagation, repeaters must have overlapping coverage for reliability. Any repeater network generates a good deal of overhead traffic in the form of acknowledgment and duplicate messages, and some form of network protocol for routing and flow control is needed. Fortunately, the implication of this protocol is readily accomplished by distributing the network control functions in the repeaters.

One design for such a network, currently in the experimental stage, indicates that a microprocessor with 3 μs cycle time and 3K 16-bit words of memory will supply the needed control functions at each repeater to support a 100 kbs throughput.[3] However, this is only a preliminary estimate and subsequent experiments may suggest more or less computing capacity.

Terminals interfaced to this network must also have the capability to perform the packet formatting and network protocol functions. It has been found that terminals with microprocessors are generally more cost-effective in terms of size, weight, and power consumption than terminals without central processor unit (CPU) power. Such hard-wired units tend to use radio channel resources inefficiently.

Several years have elapsed since the Robert's paper, and microcomputer technology has emerged from its infancy. Because of this single major innovation, the outlook for a packet radio terminal has radically changed. In this paper we reexamine, in light of today's technology and the requirements imposed by packet radio development efforts,[3-5] specifications and design issues for a digital terminal for packet broadcasting. We also discuss how these issues have influenced the design and implementation of two terminal prototypes fabricated at the University of Hawaii and SRI.

* This work was supported by the Advanced Research Projects Agency on Contract No. DAHC 15-73-C-0187 and Contract No. NAS2-6700.

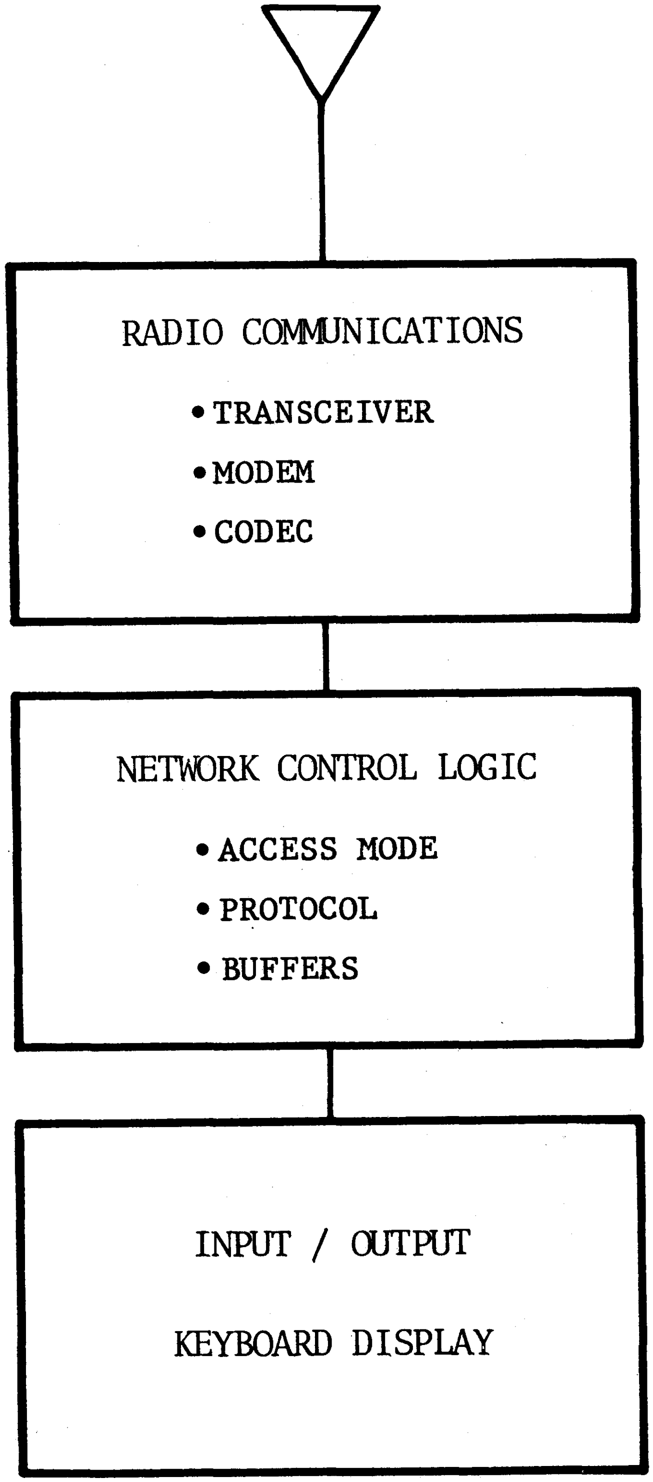


Figure 1—Functional diagram of radio terminal

FUNCTIONAL ORGANIZATION OF RADIO TERMINALS

Figure 1 is a functional diagram of a radio terminal. It is shown in three parts: radio communications (RC), network access and control logic [network control logic (NL)], and input/output (I/O). We are accustomed to thinking of terminals primarily as an I/O interface device because in a wired network the I/O is usually packaged separately and separated from the NL by very simple communications devices and long wires. However, in a radio network, the NL and RC devices must be physically adjacent to the I/O, or the mobility advantage of the radio will be lost. Hence, a radio terminal must be approached from a new point of view; it must contain a share of the NL.

The communications package in a terminal containing transceiver, modem, and codec is best designed for a specific network, since frequency, modulation, and coding may be different in each network. Thus, a single design that is able to operate in several networks would be very inefficient. In short, the RC package is network-specific and should be hard-wired.

On the other hand, network access and control logic are likely to be similar enough from one packet network to another, in terms of logical functions and required throughput, so that such functions may be efficiently implemented using a microprocessor. In fact, since a few of the net-control functions overlap in time, time-sharing a microprocessor CPU and memory may prove the most efficient approach.[6]

The I/O devices are terminal-specific and may be physically integrated with NL or separately packaged. If separately packaged, then standard I/O devices such as CRT or TTY wired-net terminals might be used; however, since no small portable I/O terminals are available, this approach must sacrifice much in mobility.

Thus, four terminal packaging configurations are possible:

(1) Separately packaged RC, NL, I/O
(2) Integrated RC, NL with separate I/O
(3) Integrated NL, I/O with separate RC
(4) Integrated RC, NL, I/O.

Each configuration has advantages and disadvantages, and each has been built for experimental purposes to verify these. We discuss the impact of these configurations on hardware and software design in the next section.

TERMINALS

ALOHA system terminal control unit

The ALOHA system[2] terminal control unit (TCU) consists of UHF antenna, transceiver, modem, and buffer. The first versions of the ALOHA TCU were packaged in configuration (1) of the previous section (i.e., RC, NL, and

I/O were packaged separately) and the total cost was $8,000 to $10,000. The next version was packaged in Configuration (2)—integrated RC, NL with separate I/O. These first versions used hard-wired logic for net-control functions, and the protocols, once set, could not be easily altered.

The most recent version of the TCU, called the Integrated Control Unit (ICU) uses INTEL 8008 and 8080 microcomputers. The ICU is completely programmable and its flexibility enables the use of a variety of different transmission protocols including variable length packets and character-by-character transmissions.

A block diagram of an ICU with an INTEL 8080 microcomputer is shown in Figure 2. The hard-wired interfaces establish synchronization and transmit bytes after converting them to bit serial form. The receiver interface performs a serial to parallel conversion and performs byte synchronization. The functions of the 8080 CPU which are performed in software are:

- Packet receive, which checks the header and text parity of an incoming packet
- Parity generation, which generates parity for both the header and text of an outgoing packet
- Packet transmit, which formats header, adds parity, sends the packet to the radio for transmission, and waits for ACK (acknowledgment) to be posted by the RCV (receive) routine.

If POSACK is not received after a certain preset interval, it sends the same packet to the radio for retransmission. After "n" tries, the routine signals a "failure to transmit." The software also contains a CRT or TTY I/O routine. The state transition diagram of an ICU program is given in Figure 3.

The evolutionary process of designing the various versions of the ALOHA TCUs has indicated that unless speed considerations dictate hard-wired logic, it is always preferable to use programmable logic. The added advantages of flexibility, ease of design, speed in implementation, and lower development costs that microprocessors provide clearly outweigh the speed advantages of hard-wired logic. One particular exception to this rule is the case of the parity encoder/decoder which needs to be implemented in hardware because the present-day microprocessors are not fast enough to meet the requirements.

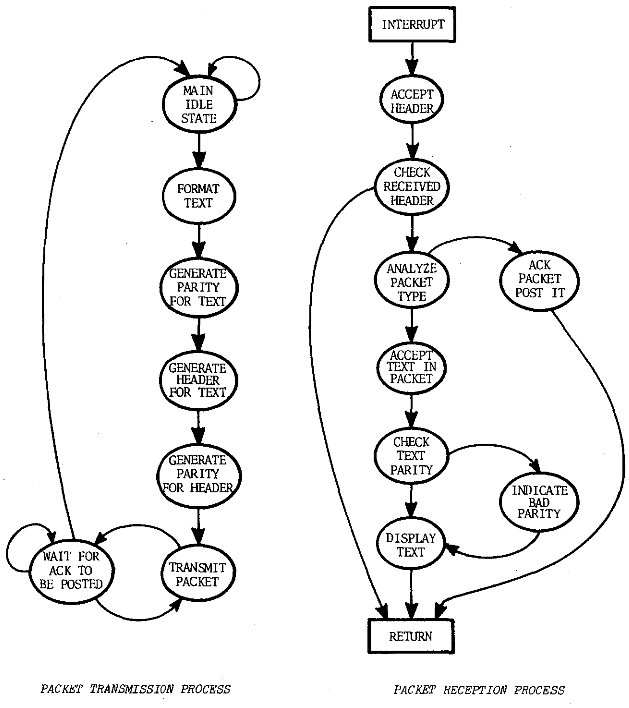


Figure 3—State transition diagram

Suitcase packet radio terminal

A different portable packet radio terminal (PRT) has been developed at SRI in conjunction with some experi-

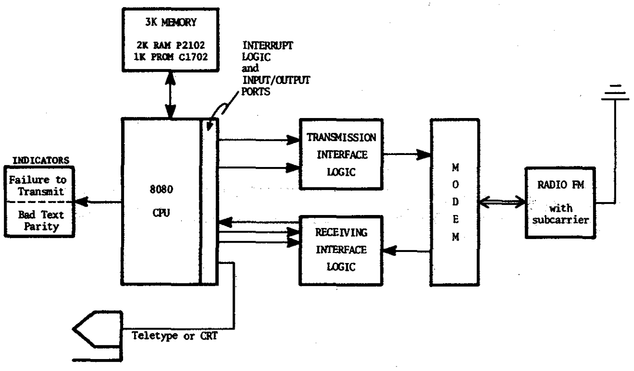


Figure 2—ICU block diagram

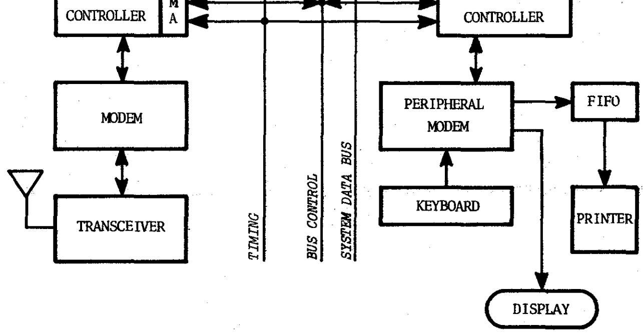


Figure 4—Terminal organization

mental traffic studies for a packet radio project. This terminal is packaged in a small suitcase with RC, NL, and I/O integrated. Parts cost approximately $5,000.

The general organization of the terminal is illustrated in Figure 4. Central to the terminal is the system data bus of the National IMP-16L Microprocessor. The CPU, peripheral controller, and modem controller all communicate with each other and with the main memory via the bus. The peripheral controller contains a buffer memory of 256 characters and controls operation of a 72-key ASCII encoded keyboard, an 80-character LED display organized in four 20-character rows, and a 20-character/line printer. The modem controller operates the modem and radio to receive and transmit packets.

In use, the operator generates a message on the keyboard in a local mode. When the carriage return key is stroked, the CPU automatically formats the message into a packet, places the packet in a transmit buffer, and passes control to the modem controller. The modem controller fetches the packet, generates parity bits, and transmits the packet. If the message is successfully received at the central computing facility, an ACK packet is transmitted to the terminal. The modem controller places the ACK, as well as all received traffic, in a receive buffer. The CPU analyzes the packet and takes the necessary action. It may retransmit when no ACK is received, or it may abort, depending on operator-specified parameters, and so forth. The user may specify whether received traffic is to be displayed, printed, or both.

Several lessons have been learned in the development of this terminal. In particular, we found that off-the-shelf microprocessor systems are not densely packaged, do not use power conservatively, and are difficult to interface. Each observation suggests that off-the-shelf microprocessor systems are not optimally suited for future terminals so that the next generation should emphasize the use of arithmetic logic unit (ALU) chips combined with microcoded read only memory (ROM) chips to tailor a microprocessor to the packet terminal.

As noted later, microprocessor technology is changing so rapidly that new devices better suited to the needs of a packet radio terminal may ultimately be available. At that time the flexibility of an off-the-shelf device may cause a change in design philosophy.

We have found that the microprocessor component should have both a bit-serial interface to exchange packets with the RC component (and characters with standard TTY type I/O devices) and a byte-parallel interface to exchange characters with integrated I/O devices. These two interfaces should be standardized for all broadcast packet networks so that terminals can be interoperated by changing the microcoded software and the RC component.

PROTOCOL IMPLICATIONS

Demands

Communications protocols, which are essential for an orderly flow of information to and from the terminal, place a heavy burden on digital terminals. Introducing a digital radio broadcast system places even greater demands on the logical capability of the terminal. This is primarily because terminals must accept traffic as it is offered; that is, there is no significant memory capability in a radio channel.

Because traffic must be accepted in an absolute on-line real-time sense, the terminal must be carefully designed around the network protocol. Thus, data rates and packet formats become crucial design elements. From a user point of view, it is essential that the radio system be transparent, that is, the user must view his terminal as a conventional time-sharing terminal. Thus, power/on-power/off functions must automatically introduce the terminal into the radio network and correspondingly indicate the terminal's departure. Acknowledgments, error control, retransmissions, and a host of other protocol issues must be imbedded in the terminal and invoked automatically.

In a sense, the protocol issues pervade the entire design of the terminal. Because buffering is related to acknowledgment procedures and utlimately to a display (or output) philosophy, it is apparent that protocol affects the organization and control of the terminal's peripherals. There are also impacts in the area of interrupt structure, keyboard interface, and so forth.

Protocol

The key protocol issues which must be addressed in a terminal include:

- Validation of ID
- ACK/text discrimination
- Duplicate packet rejection
- Error control
- Text handling (buffering)
- Transmission and retransmission logic
- Encrypting of text.

The issues of packet routing through a network of radio relays have considerable impact on terminal logic;[4] however, these issues are considered outside the scope of this paper.

To illustrate the protocol aspects of terminal organization, it is useful to examine an exemplary digital packet radio format. Figure 5 represents a typical simplified ALOHA format.

The ALOHA system staff have found that three general

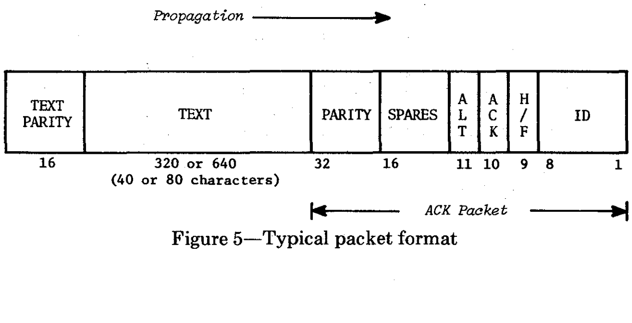


Figure 5—Typical packet format

packet formats meet their needs. These include an ACK packet (header data only) and two text packets (either 40 or 80 characters).

Inasmuch as a radio transceiver has no way of determining a priori what type of packet is being received, it is clear that it must make such a determination on the fly. Therefore, certain fields in the packet header must be searched.

The first logical check performed is to analyze the ID field to ascertain whether the terminal is the proper destination. Presuming the packet is directed to this terminal, the processsing continues. Otherwise, the receiver is reset. Parity checks are ordinarily conducted in hardware in parallel with the software-controlled tests of header fields. (This discussion assumes the packet was received with no errors.)

Given a valid ID, the terminal must then check the type of packet. In our example, this means examining Bit No. 10. If the bit is set, indicating an ACK packet, the terminal must check its transmission buffer to verify whether a recently sent packet is awaiting acknowledgment. If so, the transmission buffer and the receive buffer occupied by the ACK are released.

If the packet is not an ACK, it is assumed to be text (in our example). In this case, the ALT Bit (Bit No. 11) is checked to reduce the probability of receiving the same packet twice since this bit is complemented every time a new packet is sent to the terminal. If the terminal fails to acknowledge receipt (or the ACK is not received at the sender), the packet is resent with the ALT bit unchanged. In the case of a duplicate packet, it is ignored; the receive buffer is released, and the ACK transmission logic is exercised again. Note that this particular discussion is idealized—the ALOHA terminals do not currently acknowledge received traffic and this discussion applies only to traffic received at the station.

Assuming a valid text packet is arriving, the terminal then checks Bit No. 9 to determine whether it is a 40- or 80-character packet. This usually requires setting a hardware counter in the modem interface so that text parity is checked properly.

Presuming all of this logic is satisfied, the terminal is then free to output the packet at its leisure.

In our particular example, it is possible for several errors to occur. A packet may be received with parity errors in either the header, text, or both. These errors can be monitored or ignored since the terminal has the option of accepting the packet or immediately resetting. From an experimental point of view, it makes sense to monitor errors since the channel is effectively blocked for the packet duration.

If errors are monitored, the station is frequently used to send control packets to each terminal for error counts to be broadcast back to the station.

Transmission logic for a broadcast channel is limited but not particularly complex. Protocol demands that a transmitted packet be saved until an ACK is received from the destination. If an ACK is not received in a predetermined time, the typical protocol dictates retransmission in a pseudo-random time interval. Pseudo-random times are selected to reduce the probability of several terminals jamming each other repeatedly while competing for the channel. In the ALOHA system, a packet may be retransmitted up to five to eight times before the terminal gives up and notifies the user. Inasmuch as most time-sharing system users require their traffic to arrive in sequence, it is common to have only one or two transmission buffers and to lock the keyboard when the buffers are filled.

Software

Terminal control

Terminal control is readily exercised through a microprocessor supervisor or executive program. Because of the real-time demand of the radio channels and the less constrained output demands, it is convenient to think of the software as being organized in a foreground and background mode.

In such a "multiprogrammed environment," the foreground partition is interrupt driven to accommodate the modem interface and its real-time demands. Such a partition must have first priority and if the software is organized properly, the response time to interrupts can satisfy most data rate requirements.

In our terminal, the data transfer between modem interface and memory is performed under DMA control. Since data are transferred on a word by word basis, the processor has more time to react, and even slow microprocessors can accommodate large rates (in excess of 100 kbs).

The foreground partition is generally devoted to the receive logic and receive buffer control. Certain modem-dependent transmit code must also reside in the foreground; however, the general transmit logic (e.g., retransmission timing, and so on) is less time dependent and therefore can reside in the background partition.

The background partition ordinarily contains the terminal's peripheral control routines. Such activities as display, edit, format, print (if appropriate), keyboard interface, and so forth, are monitored in the background. Data transfer between peripherals and the CPU is most conveniently handled under CPU control so that the hardware interface is simpler and standard programming techniques can be used.

Interrupt polling

The interrupt structures and options provided by microprocessor vendors are varied. In the suitcase terminal we have used a National Semiconductor IMP 16/L CPU. This machine has four interrupt levels. One level is vectored and is an obvious choice for responding to the modem interface.

Software development

Software development for digital radio terminals is limited by the constraints inherent in microprocessors. There are not only limitations in the instruction sets but also memory problems, speed conflicts, and poor (in general) programming aids.

Programming aids

In practice, microprocessor vendors provide both resident-assemblers and cross-assemblers. In the IMP-16L case, National Semiconductor provides a cross-assembler for the IBM 360. INTEL provides an algebraic processor in addition to assemblers.

The National Semiconductor resident-assembler is provided in object form on paper tape. Approximately 25 minutes are required to load the assembler. As a three-pass assembler, it requires entering the source code three times. An inherent disadvantage in the National Semiconductor software is the inability to load the loader and assembler in memory simultaneously. Thus, the constant loading and reloading compounds the debugging problems considerably.

However, the cross-assembler can be a significant program development tool. The National Semiconductor cross-assembler is supplied in source FORTRAN IV for an IBM 360. SRI modified this code extensively and implemented it on a DEC PDP-11/20 with 28K words and a Vector General Display System. This system provides high speed paper tape facilities and an extremely powerful editing system. In addition, it was very successful in providing hands-on debugging at the source level. An emulator would be even more beneficial.

Our software experiences with microprocessors were not surprising. They are, indeed, much less sophisticated than minicomputers and the software support from the vendor is limited—at best. Our experience with the IMP-16L software was very unsatisfactory as supplied; however, our investment in the cross-assembler was very worthwhile. Furthermore, we would expect similar quality in software supplied for any new digital processor—micro, mini, or large main frame.

TECHNOLOGICAL CONSIDERATIONS

Physical characteristics

The natural evolution of packet broadcast terminals has been toward greater portability and lower cost. Initially, terminals packaged in Configuration (1) were not portable, and cost $8,000 to $10,000. Currently, the ICU occupies .6 cubic feet; weighs 15 lb without keyboard, display or battery; and costs $2,000 in parts. The battery pack, including charger, is the size and weight of an automotive battery. Although the latter two terminals were developed with the intention of portability, little attempt has been made to minimize their size or weight.

To realize the full potential of packet broadcasting, future design efforts must concentrate on achieving a single physical package containing RC, NL, I/O, and power supply. The I/O must be engineered from a human factors viewpoint to be convenient, easily learned and operated, and must be designed to avoid operator fatigue. The self-contained power supply should provide a minimum of four hours continuous operation and should be readily recharged or inexpensively replaced. Finally, the entire package should be as small and light as possible consistent with other objectives. In this section we discuss the possibility of applying existing technology to achieve the goals and discuss where advances are needed.

Input/Output

The I/O elements interface the man to the network. If they are poorly conceived or implemented, the best technological design of all other network elements cannot compensate for these deficiencies. In this paper we assume that the input element is a keyboard, because that seems the most likely initial component. Subsequent study may show that some other forms of input (such as hand-written characters or spoken words) are preferable. Similarly, we have assumed an alphanumeric display as the most likely initial candidate, although subsequent study may not support this assumption.

Displays

The important physical characteristics of the display include character clarity, size, color, and contrast; display format; power consumption; and overall size. The trend of portable display technology seems to be toward 5×7 or 7×9 dot matrix characters between 0.1 and 0.2 in. high, although nine and fourteen-segment displays are available. Most displays (LED or Plasma) are red on black with some yellow and green displays now on the market. Liquid crystal display color depends on ambient light, and virtually any color is possible. Since character size will determine the maximum total number of characters displayed, a determination of minimum acceptable character size is very important. A human factors study of character size, color, and contrast for portable terminals would help greatly in design of a suitable display; however, pending the results of such a study early terminals will depend more on selections of available display devices.

The display format most desirable is probably a 12- to 16-line page with 72 or 80 characters per line. Such a display may eventually be possible with 0.1 in. characters on a 3×8 in. area; however, except for CRT displays, this density is not available today. If the maximum display dimension is limited to 8 in., then off-the-shelf technology of either LED or plasma displays limits the number of characters per line to 40. Using some advanced LED technology not yet in production, it would be possible to construct a display of twelve 40-character lines in an 8×3 in. area; however, the cost and power consumption would be excessive.

The normal dot on a standard LED matrix requires approximately 30 mW of input power. Assuming 20 active dots per character, a 40-character line would require 24 watts. To be consistent with desired size and weight properties, the display should consume no more than one or two watts. Thus, a single-line 40-character dot matrix LED display would require a 24-fold improvement in efficiency, and it is not likely that a multiple-line dot matrix LED display will ever be satisfactory. Other LED configurations are possible, however, both 9-segment and 14-segment character fonts are available. If an average of 5 segments is needed to display a character, the power requirement would be only ¼ that of a dot-matrix.

It is possible to use a CRT display as an interim solution. A 4 in. CRT will display 12 lines of 40 characters in very readable fashion. Such a display can be packaged in a 250 cubic in., 8 lb, 11 watts package. However, it is not likely that any significant size, weight, or power reductions are possible, so the CRT is not a promising solution.

Liquid crystal matrix displays recently announced by Hughes[7] and Hitachi[8] will approach CRT dot density at low power-consumption, and may provide the desired full-page display; however, these displays are not yet in production, and no detailed specifications have been published.

A great deal of research effort is being directed toward alphanumeric displays, with goals of obtaining higher character density and lower power. This brief description was not intended to be a survey, but to indicate that the display requirement for a packet broadcasting terminal, though severe, will soon be met.

Keyboards

In a sense, the keyboard is a more difficult problem than the display, since it must provide the ability to enter any one of a large number of characters. Furthermore, it must be arranged so that a large finger or fat thumb will depress only a single key.

Many approaches are possible. Roberts proposed the use of a binary encoded 5-key device developed and used at SRI.[9] Although this device may be operated almost as rapidly and mastered more quickly than touch-typing, it has the disadvantage of many specialized computer devices that must be learned. Uncoded keyboards (every key is a separate character) have the disadvantage that the number of keys must equal the size of the character set; however, a novice can use an uncoded keyboard with no instruction. A reasonable compromise is a partially encoded, or multifunction keyboard such as that used on most hand calculators. These keyboards can be made self-explanatory so that a novice can use one by examining the labels of the keys. If at all possible, a standard TTY layout of the alphabetic keys should be used with the interkey spacing and switch "feel" as similar to TTY keyboards as possible. A possible keyboard organization would divide the character set into three subsets: uppercase (standard data entry); number symbols; and control characters (TTY control set). Two shift keys, a space bar, and an "enter" key are also needed. Special functions such as character or line delete can be encoded as part of the number/symbol case. Such a keyboard, with standard spacing, will occupy only 8×4 in. and will be almost as easily used as the TTY or typewriter keyboard. An example of such a keyboard with a nine-segment, 32-character LED display is a terminal manufactured by MICON Inc., for communications use by the deaf.

To obtain suitable contact pressure and overtravel, a standard set of keyswitches could be used; however, many other technologies offer thinner keyboards with slight loss of "feel." Although these cannot all be reviewed here, the conductive elastomer keyboard can be considered as representative. Such a keyboard can be designed with 0.8 in. travel and fitted with a silicon cover to provide an impression of overtravel. It is only 0.25 in. thick (including the silicon), and has no holes or cracks for entry of dirt or water; this is a very desirable feature in a portable terminal.

Network control logic components

An examination of the internal hardware design problems leads to the conclusion that to eliminate extra circuitry, all inputs and outputs to the network control logic should be handled in their natural form, and control functions should be centralized in a microprocessor.

A wide variety and number of microprocessors are available today, and technology is changing so rapidly that each month a number of new devices appear on the market. Although microprocessors were originally introduced by semiconductor houses to help sell memory, they are also sold as complete systems, nominally for prototyping new products.

The microprocessors available vary greatly in speed, number of bits per CPU, number of chips per CPU, type of instruction set, and power requirements. Compared to minicomputers, today's MOS microprocessors are limited in all performance factors; however, recently announced or planned microprocessors which use SOS, IIL, bipolar, and low-power Schottky technology are rapidly approaching minicomputer performance in all parameters. A sampling—by no means inclusive—of available and announced microprocessor chip-sets and chips is given in Table I. This table is not complete, but is provided only to show the wide variety of performance soon to be available. Reference 10 can be consulted for more complete information.

Since the radio I/O logic is naturally serial it should be handled by the central microprocessor in that form. The microprocessor definitely must have the power to do serial to parallel or parallel to serial conversion to satisfy other requirements so these functions should be centralized.

Centralizing the conversion processes also allows the microprocessor to take on the burden of packet synchroni-

TABLE I—Microprocessors

Manufacturer and Identification	Semiconductor Technology	Instruction Fetch Cycle (μs)	Available 1974
Fairchild PPS-25	NMOS*	62.5	—
RCA COSMAC	CMOS†	3.0	No
Intel 8080	NMOS*	0.5	Yes
Intersil IM 6100	CMOS†	0.5	—
T.I.	IIL‡	0.5	—
Inselek	SOS§	0.3	No
Raytheon RP-16	Bipolar	0.2	—
Monolithic Memories 6701	Lowpower Schottky**	0.15	—
Intel 3000 Series	Bipolar	0.12	Yes

* N-channel metal oxide silicon.
† Complementary—metal oxide silicon.
‡ Integrated injection logic.
§ Silicon on sapphire.
** Low-Power Schottky.

zation. All packet synchronization and parity checks should be accomplished in microcode. Bit synchronization must be accomplished in the modem. Encoding and decoding processes are independent of other functions and should be incorporated in hybrid circuitry external to the microprocessor and modem modules.

Other I/O functions should be accomplished in parallel to take advantage of their natural form. Data to be displayed should be output in parallel form. The display itself could be addressed as part of the microcode memory space or by a separate register and bus using the microprocessor hardware. The data should be decoded from standard ASCII on the data bus through a row-column generator. The display itself should be refreshed by microcode during the idle state of the microprocessor. When the processor is busy checking parity or transmitting, the display could probably be blanked for short time intervals without affecting the user. Blanking the screen could be used to notify the user that his packet had been transmitted or that a new packet had arrived and was in the terminal. Keyboard data should be encoded into an 8-bit code and input in parallel, since the keyboard lends itself to a parallel format and the data bus will be 8 bits wide. Assuming a 32-key multifunction keyboard, such as described in Section V, is used, 5 bits can be used for the 32 keys, and the other three bits for the "enter" and "shift" keys. A sample coding format is shown in Figure 6.

This formatting allows quick table lookup in microcode to translate to ASCII from keycode input using a 7-bit address.

An initial conservative estimate of memory size for the microprocessor is 256 bytes of read/write RAM for buffering and 2K words of 16-bit ROM for microcode. These estimates are based on the memory requirements of the ICU modified to compensate for the microcode type of operation and a 128-character display size. For example, a pair of Intel 8316 MOS ROM may make an attractive circuit package for holding the microcode. The circuit, organized 2048×8 bits, has a low power dissipation of 10.7 μW/bit and runs from a single 5 volt supply. This pair of ROM packages provides the required 2K words of microcode using two 24-pin spaces. Although the masking charges are expensive, a standard ROM package to hold microcode has advantages over a special CROM package which contains control logic for the microprocessor. A writable microcode memory external to the package can be substituted for the ROM for testing and microprogram development.

Judging by the current rate of change in the industry, suitable microprocessors will probably be available in a year or two; however, to meet the power and size requirements in the interim the microprocessor element probably must be custom designed. To restrict power consumption it will probably be necessary to construct the unit using hybrid techniques and low-power Schottky MSI devices. Speed is extremely important because of the serial interface to the radio; however, fancy microinstructions are not. The microprocessor needs only the primitive operations, such as AND, OR, XOR, RIGHT-SHIFT, ADD, COMPLEMENT, and INCREMENT, plus a few positive and negative branch type instructions. It must also have some internal routing microinstructions. Experience with the ICU indicates that microinstructions should execute on the order of 200 ns/instruction. Typical power consumptions for ALU and microcomputer integrated circuits are shown in Table II.

The complexity and speed vary quite drastically from the 74LS181 which can perform one of 32 operations on two 4-bit wide binary numbers in 25 ns to the 8008 Intel CPU which can perform an 8-bit ALU operation in 20 μs. To achieve adequate speed performance, a low-power Schottky implementation is probably necessary. Such a unit with memory should consume no more than one watt.

Radio communications components

Transceiver technology is available to allow very small, low-power packages; however, it must be applied to the specific modulation and coding design for packet broadcasting. The Motorola Dynatac[11] terminal contains a transceiver and digital modem which would satisfy the ALOHA requirements as to bit rate and transmitter power. The Dynatac package occupies 60 cubic in. and includes touch-tone pad, headset, audio circuitry, control logic, and batteries. It seems likely that the transceiver and modem portion occupy no more than 10 cubic in. Other efforts are under way to apply thin film hybrid technology to miniaturize packet broadcasting RF

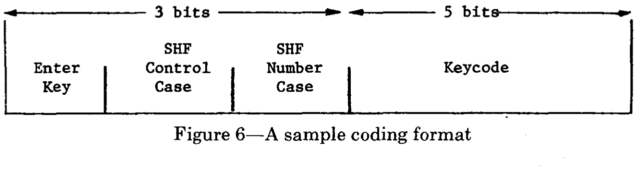


Figure 6—A sample coding format

TABLE II—Power Consumption

Technology		Integrated Circuit	Power/Circuit
MOS	4004	Intel CPU 4 bit	420 mW
MOS	8008	Intel CPU 8 bit	420 mW
MOS	8080	Intel CPU 8 bit	1 W
BIP	3001	Microprogram control	900 mW
BIP	3002	Central processing element	750 mW
LS	74LS181	Arithmetic logic unit	125 mW

and modem circuitry. These promise to achieve required performance in a 10 cubic in. package with receiver power consumption below one watt.

Although the transmitter peak power is nominally 10 watts, the duty cycle will be very slow so that the transmitter will require only a few milliwatts average power. The power source must be able to supply this low average power in short 10-watt bursts.

Power source considerations

The power source will most probably dominate the other components in determining the size and weight of a packet broadcast terminal. The ICU, containing no keyboard or display, requires 15 watts. The suitcase terminal, including an 80-character display and full ASCII keyboard, requires an additional 33 watts. Although components were carefully selected to conserve power, no attempt was made to go beyond components available off-the-shelf.

As discussed above, it is probable that a terminal can be developed which will require no more than 5 to 10 watts, with the display being the unknown factor.

Batteries are available in power densities varying from 0.5 Whr/cubic in. and 10 Whr/lb to 5 Whr/cubic in. and 100 Whr/lb. Assuming that inexpensive rechargeable batteries will be used, a nominal density of 20 Whr/lb and 1 Whr/cubic in. are possible, so that a 20 Whr battery pack will weigh one pound and occupy 20 cubic in.

With this battery, the terminal will provide from two to seven hours continuous operation depending on the display power required.

Regulation of battery-supplied power can be power consuming if close voltage tolerances are required. Selection of logic families and circuit design that are tolerant of voltage variation are major design considerations.

CONCLUSION

With today's technology, a small lightweight personal terminal is within the state of the art. The display is the only unsolved problem; however, as the liquid crystal dot matrix displays recently announced are brought to production, that problem will disappear. Suitable efforts concentrated on developing an RF hybrid package, a microcoded microprocessor, and packaging the entire unit should result in a terminal which occupies no more than 100 cubic in., weighs less than 5 lb, and costs on the order of $3,000. Quantity production can reduce this cost drastically.

Although we have only discussed possible alphanumeric terminals, future technology will make other types possible.

Using packet broadcasting technology, it will be possible to make very simple one-way terminals to either send or receive messages. Transmit-only terminals may find application in monitoring remote sensors such as weather metering instruments or the state of traffic at a busy intersection. Receive-only terminals may be used to change traffic signals or possibly to control remote advertising signs.

Current efforts to digitize speech may result in very compact, low-power speech digitizers that could be combined with packet broadcasting technology to provide hand-held terminals with both direct voice communication and data I/O using remote word recognition at the central computing station.

To understand the operational context under which the ALOHA and suitcase terminals were developed, please refer to the other papers on packet radio in these proceedings.

ACKNOWLEDGMENT

The authors wish to thank the members of the Advanced Research Project Agency research team designated as the Packet Radio Working Group who have contributed ideas, concepts and specific design details to the work discussed in this paper.

REFERENCES

1. Roberts, L. G., "Extension of Packet Communication Technology to a Hand-Held Personal Terminal," *Proceedings of AFIPS, 1972 Spring Joint Computer Conference*, Vol. 40, pp. 295-298.
2. Abramson, N., The ALOHA System, "Another Alternative for Computer Communications," *Proceedings of AFIPS 1970 Fall Joint Computer Conference*, Vol. 37, pp. 281-285.
3. Fralick, S. C. and J. C. Garrett, "Technological Considerations for Packet Radio Networks," National Computer Conference 1975, Anaheim, California, *Proceedings of this conference.*
4. Frank, H., I. Gitman, and R. Van Slyke, "Packet Radio System: Network Considerations," National Computer Conference 1975, Anaheim, California, *Proceedings of this conference.*
5. Burchfiel, J., R. Tomlinson, and M. Beeler, "Functions and Structure of a Packet Radio Station," National Computer Conference 1975, Anaheim, California, *Proceedings of this conference.*
6. Fralick, S., and D. Brandin, "The Role of Microprocessors in High Speed Portable Data Communications Terminals," *Proceedings of Journees d'Electronique*, Lausanne, SW, 1974.
7. "Developments," *Computer Design*, March 1974, p. 42.
8. "International Newsletter," *Electronics*, October 3, 1974.
9. Engelbart, D. E. and W. K. English, "A Research Center for Augmenting Human Intellect," *AFIPS Conference Proceedings*, Vol. 33, p. 397, 1968.
10. Schmid, Hermann, "Monolithic Processors," *Computer Design*; Vol. 13, No. 10, October 1974.
11. *The Dynatac Concept and the 900 MHZ Mobile Radio Band*, Motorola Technical Report submitted to the FCC related to docket No. 18262, April 1973.

Area Director:
Glyn H. Jones
Burroughs Corporation
Mission Viejo, California

Software

The software technical sessions were conceived and planned so that the 1975 NCC sessions on software concentrate upon a number of fundamental issues related to software. The primary objective is to illustrate how systems designers can provide products that truly serve the needs of the people, and the emphasis is on *people*; we also wish to examine techniques which can yield a definitive measure of software quality, and to discuss some of the key issues involved in creating improved software and software development tools.

We have acquired the services of many prominent leaders in the industry to act as session chairpersons. Together they formed a strong planning team.

Research and development on computer software is presently concerned with the development of tools and techniques which can lead to the cost-effective specification and production of reliable computer programs, with the emphasis on reliable! All of the sessions are directed at the effective utilization of human resources; from the standpoint of overall system design, management of programming, the practice of programming. The sessions present many facets of the current research and development on computer software.

The first panel discussion "Programming is an Act of Communication" will provide some thought-provoking ideas to be pondered during the following sessions. Some of the topics related to program portability, the principles of unique definition and software quality will be treated in sessions examining the need for precise thinking about system design.

Techniques for formally verifying the proper operation of computer programs and the development of tools to evaluate the effectiveness with which computer programs utilize the available hardware resources will be highlighted in the session "Program Verification in 1980."

Building on the work of others is the only way to make substantial progress in any field. Increased productivity of programming teams is achieved with the development of a better style in programming, effective use of modularization, by an understanding of the issues involved in designing new programming languages and consideration of language extensibility. These topics will be explored in the sessions "Issues in Programming Language Design" and "Programming—Art, Science or Engineering?"

The recent significant improvements in Program Management, particularly those concerning design reviews, obtaining and analyzing empirical data on software engineering and improving the support of existing program products will be discussed in the session "Software Engineering."

The session "Operating Systems" shows the means by which innovative advances in modelling of systems and mechanisms for redefining virtual machines can lead to more powerful computing tools.

Finally, a session on "COBOL 74" examines this new language in some detail, giving emphasis to its suitability for software engineering.

The software sessions emphasize real problems and operational solutions to these problems so that attendees can learn about advances which will be useful to them in their current assignments.

On the principle of unique definition

by P. D. GRIEM, JR.
The Foxboro Company
Foxboro, Massachusetts

THE PRINCIPLE

The principle of Unique Definition can be stated as follows: "Unique definition of software constructs tends to increase software reliability."

To many people, this principle will be so obvious that it hardly deserves a formal statement, much less an entire paper. Since a software construct can have only one correct definition, then if additional definitions exist, they must either be identical to the correct definition, and hence superfluous, or different from the correct definition, and hence erroneous.

Yet, as will be brought out later, current software is fraught with multiply-defined constructs. This paper discusses:

- why multiple definitions of a construct decrease software reliability
- tools which aid in constructing unique definitions
- how Unique Definition was applied to the design of a software system for industrial process control, and the resulting software modularity
- some notes for applying the principle.

DANGERS OF MULTIPLE DEFINITION

As used in this paper, "programming" means all the work necessary to define software to a programming system so that it can produce machine-executable code. Thus "programming" is used very broadly, and includes not just the writing of source statements in a programming language such as Fortran, but also any necessary compiler directives, file definition statements, Linkage Editor commands, Job Control Language, etc., *as well as* the mechanical processes for entering these definitions into the programming system. These mechanical processes, especially where humans are involved, often provide more opportunities for error than the program-writing processes.

When defining software (programming), the programmer employs various software constructs, e.g., program logic structures, data structures, overlays, syntax analyzers, etc. Some of these constructs may be well defined within the programming system, and need only be referenced or invoked by the programmer for the programming system to create the desired software result. Other constructs needed by the programmer may be partially or entirely undefined, thus requiring him to define them (and giving rise to complaints about "reinventing the wheel," software "transportability," lack of "natural" languages).

In the past, many, if not most, of the constructs employed and created by programmers were poorly understood. Today, after many doses of the medicine prescribed by Dr. Dijkstra[1] and others, we are beginning to appreciate software structure and basic constructs.

But recognizing software constructs is very difficult. Though programming systems may improve in this respect, they will never anticipate all the constructs needed to solve application problems; programmers will always be defining software constructs to programming systems. The errors in that process can be minimized by following the principle of Unique Definition: define the construct once, and thereafter reference that definition.

If a programming system requires multiple definitions of some construct (or strongly discourages unique definitions, which has the same effect), then the programmer is presented with multiple opportunities for error. The total possibility for error must be some function of:

- the intellectual difficulty of formulating the definition,
- the number of definitions required,
- the number of different ways in which the definition must be stated,
- the mechanics of entering the definition into the programming system, and
- the number of people involved in the total programming process.

As the opportunities for error increase, it is not long before error is virtually guaranteed. ("To err is human . . .")

At this point, it would probably be helpful to give a few current examples of multiple definition:

A. A basic programming construct is the DO loop; some form of it is defined in many high-level languages. The programmer may invoke it by a simple statement, e.g.,

$$DO\ 10\ I=1,15,2 \qquad \text{(Fortran)}$$

However, the DO construct is absent from assembly language. The programmer must build (i.e., define) it himself each time from more primitive statements, choosing from a wide variety of possible ways.

B. To use a COMMON block in Fortran, each program or subprogram must contain a complete definition of its format.

C. To create overlays, some programming systems require the programmer to state the association of a particular subprogram name and a particular overlay segment name on each call to that subprogram, in addition to a separate statement of that association in different language to the Linkage Editor.

D. During the generation of the operating system software for some machines, it is necessary to state several times in several different ways that the machine has, say, five tape drives.

E. Most programming systems provide no way to define packed data. The programmer in effect defines the attributes of a packed data field by the (different) code necessary to extract or insert a value.[2]

F. On some programming systems, to generate machine-executable code from program source statements requires an operator at some terminal to invoke separately and give commands to a compiler, assembler, link editor, and file utility program. The entire process must be repeated whenever the program is to be regenerated. On more sophisticated systems, these invocations and commands can be entered into a file once, to be executed by a Job Processor whenever the file is referenced.

G. Each reentrant program may require special coding, especially if written in Assembler language, as compared to a non-reentrant program.[5]

Many other examples could be given, but it should be clear that just writing multiple definitions of the same construct affords multiple opportunities for error. Worse still, multiple definitions are also difficult to manage intellectually, in the following ways:

1. Creating multiple definitions of a construct is not an intellectually stimulating task.
2. If the multiple definitions are coded in different ways, the programmer in reading the various statements cannot by simple comparison determine whether all the definitions are precisely equivalent—instead he must make an intellectual effort to translate the definitions into a common means of expression, and then compare them.
3. Just keeping track of all the definitions is a significant mental effort. When the number of definitions of a construct is small, the programmer can feel confident and in control of the program. As the definitions (and particularly the opportunities for error) multiply, he becomes increasingly uneasy and unsure. Finally, at some point, he feels that he can never be certain that all definitions of the construct are correct. Some programmers cannot accommodate this final mental state, and must either reorganize the programs ("it's a complete mess—the whole thing will just have to be rewritten") or change job assignments.

Another problem is establishing or changing all the definitions of a construct. Usually it is impossible to physically change all of them simultaneously, so during the time required to change the definitions the software must contain conflicting definitions. And there is always the danger that the process of change may not be carried out correctly, or that it may be interrupted and never completed.

TOOLS FOR UNIQUE DEFINITION

Since the programmer must define constructs and reference them, a programming system can be evaluated on the basis of the tools it provides for doing so. A few current tools come to mind immediately:

1. Subroutines or procedures allow the unique definition of programmed functions, and are a very common feature of programming languages.
2. Macros allow the unique definition of coding structure, and could be used to create an assembler DO statement for Example A above. Macro capability is a common feature of assemblers, but strangely is seldom found in processors for high-level languages. Language-independent macroprocessors exist,[9,10] but are rarely available to, or understood by, the average programmer.
3. The INCLUDE statement in the preprocessor of some PL/I compilers, copies the contents of a file (which can contain any arbitrary text) into the source text of a program being compiled. This allows the unique definition of statements which must appear identically in several programs, usually data and format declarations; it would solve the Fortran COMMON problem of Example B above. The INCLUDE statement is a very powerful tool, occasionally present in programming systems for large machines. A related concept is COMPOOL.[7]
4. Libraries allow the unique storage and retrieval of, for example, subroutines, macros, source text, data files, etc., according to the type of library.
5. Symbol cross-reference (concordance) listings show where definitions are made and referenced. A common feature of assemblers, it is rarely found in other programming tools, except sometimes in compilers for large machines.

Such tools, while valuable, are not adequate to solve Examples C, D, E, and F above. More sophisticated text processing might be one general approach. In some cases, the best solution might be a basic modification to the programming system, which is clearly needed for Examples E and F, or to the operating system—virtual memory removes the concern for overlays in Example C.

A REAL-LIFE PROBLEM

Very briefly, I will try to describe how Unique Definition was used in the design of IMPAC, a large software system (>200K source statements) for industrial process control.

Description

IMPAC consists of program and data modules, mostly resident on bulk memory, which run under control of a real-time multiprogramming executive on the FOX 1 process computer. Typical of such systems, IMPAC employs the concept of a Block (Figure 1). Each Block may receive one or more inputs from the process or other Blocks, calculate some simple algorithm, check values against alarm limits, and produce one or more outputs to the process or other Blocks. Blocks of various types may thus be connected to the process and other Blocks to form a control system, for example, for a petroleum distillation column. Hundreds of Blocks might be implemented on one computer.

Process control software such as IMPAC must provide some means of generating Blocks to form a control system, "executing" the Blocks to operate the control system, and communicating between the control system and the plant operator, the plant engineer, and application software. A common approach has been to establish a Block File, with each record containing the information necessary to represent one Block. The record length and format varies according to the Block type; a record may consist of 10 to 100 fields (mostly packed) which contain the description, parameters, status, working storage of the Block, often in coded form. Communication with the control system mainly consists of manipulating the values contained in the fields of its Blocks.

At the time that the IMPAC software design was begun (1970), the conventional design for this kind of software was to build, for each function mentioned above, a separate program which directly accessed the fields of the Block records,[8] as shown in Figure 2. Thus, each program in effect defined the fields of the Block records in detail, by the coding which manipulated those fields. As a result, most fields were defined multiply and differently (because of different program designs). So it was difficult to write correct software initially, and harder still to modify it later for development or application purposes.

It was clear early in the design of IMPAC that these problems could be reduced if the field definitions were uniquely defined, stored in a file, and consulted as required by the IMPAC or application programs through some convenient access mechanism (Figure 3). The access mechanism could also handle the transfer of data between the Block File and any program as well as resolve any contention problems. To do this, the characteristics of the definition of a field of a Block record had to be determined. These were found to be:

C1. The function of the field in the calculations performed during the "execution" of the Block.

C2. The location of the field within the Block record, usually packed (Record format).

C3. The unpacked form used for computation and manipulation by IMPAC and application programs (Internal format), i.e., the data type.

C4. The ASCII form used for display on, or data entry from, devices external to the computer (External format).

C5. The conditions under which the value in a field may be changed by programs other than the Execute program.

A major problem was how to preserve unique definition and still allow software to be built for good efficiency when necessary. The Execute program is the major

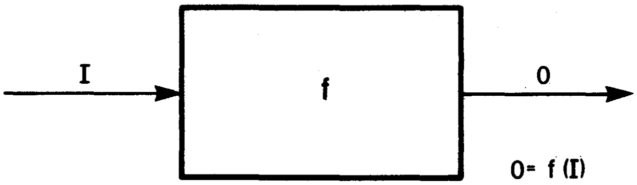


Figure 1—A "block"

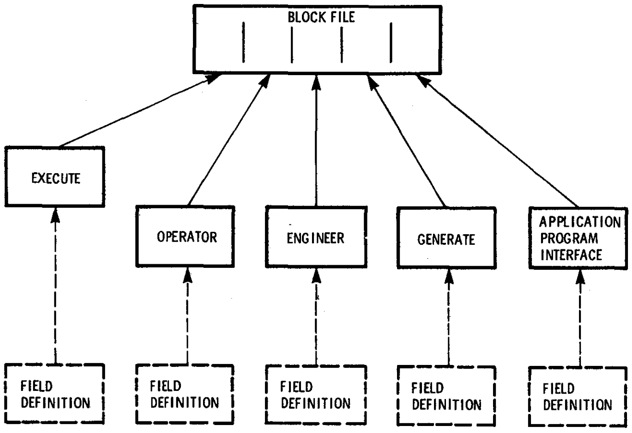


Figure 2—Conventional software design

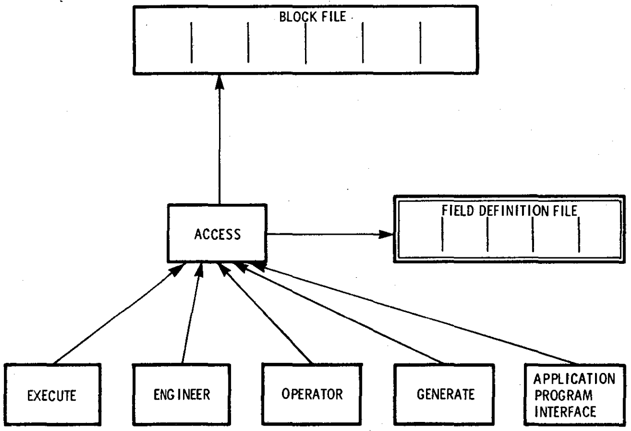


Figure 3—Ideal software design

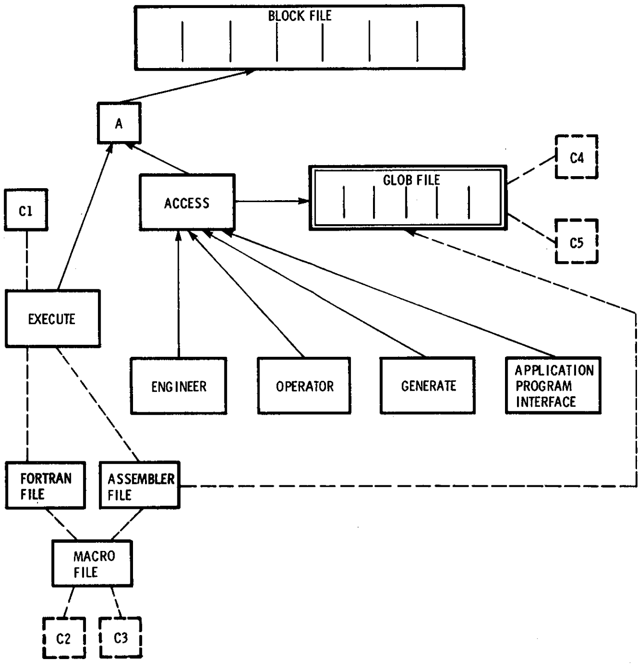


Figure 4—IMPAC design

activity in the computer; it interpretively executes up to a few hundred Blocks per second, accessing dozens of fields per Block, so the run-time overhead of calling an access mechanism for each field would be intolerable. In general, for the other programs, references to the field definitions can be resolved at run-time, but in certain cases better operating efficiency is also needed for these programs.

Examination showed that the Execute program had to reference only the definitions of characteristics C1-C3, and that these should be physically resident in the Execute program in the form of in-line code for best efficiency. All other programs had to reference definitions of characteristics C2-C5; and these were encoded for each field into an information block called a GLOB, with all GLOBS stored in a file (see Appendix A for a more detailed description of a GLOB). A program called ACCESS provides functions to get the value(s) of the specified field(s) from a specified Block record and return them to the calling program in Internal or External format by referencing the appropriate GLOB(s), and vice versa, to give new values in either format. Also, ACCESS resolves contention between programs which call it, and a submodule of ACCESS resolves contention between ACCESS and the Execute program for acquiring Block records.

Since definitions of Record format (C2) and Internal format (C3) must physically reside in the coding of both the Execute program and the GLOBs, the definitions must be made external to both codings but referenceable by both. This was accomplished as shown in Figure 4, and as described below:

a. The definitions of Record and Internal format are stated in a macro language.
b. A macroprocessor, MAX, is used to create files of equivalent Fortran and Assembler language packed data definition statements.[2]
c. The source code for the GLOBs and the Execute program references the files via INCLUDE directives; the language processors read the packed data definition statements and create the most efficient machine code possible whenever a conversion between Record and Internal format is necessary.

As a result, the system provides three choices of operating efficiency for programs communicating with IMPAC Blocks:

1. By calls to ACCESS, where ACCESS must fetch the required GLOB(s) from the GLOB file at run time. This method has the longest execution time (because of GLOB file accessing), but is the most strongly encouraged, since the field definitions (i.e., GLOBS) may be modified without modifying or regenerating the calling programs. All IMPAC programs except the Execute program use this method.
2. By calls to ACCESS, where ACCESS uses a GLOB in the calling program; the GLOB was extracted from the GLOB file and incorporated in the program load module at Link Edit time. This method is seldom used by IMPAC programs since they do not usually know until run time which GLOBs they may have to reference, but is widely used by application programs since they do.
3. By in-line code which resulted from referencing the Assembler or Fortran definitions through INCLUDE statements. This method was intended for use by the Execute program only, but was found necessary in a few other IMPAC programs for basic fields such as Block name and Block record size. It is the most efficient, since there are no ACCESS calls, but it is the least encouraged, since complete program regeneration is necessary if a referenced definition changes.

This design approach has effectively decoupled data field properties from program logic. Modifying the definition of a field requires changing only the code which expresses that definition, followed by some amount of program regeneration. But it should be noted that to achieve this state requires a programming system containing language processors with somewhat unusual features (INCLUDE and packed data), a macroprocessor, and a file library system.

It should also be noted that this system does not really provide unique definition of the data type (C3) of a field. Since Assembler and Fortran program coding is different for different data types, any program which deals with a field in Internal format must know (and thus carry a definition of) the data type of the field. Fortunately, it has never been necessary to change the data type of a field. Also, the function (C1) of a field can be uniquely defined to the IMPAC software only; an application program which references a particular field usually, but not always, knows (i.e., redefines) its function—this seems to be unavoidable.

MODULARITY

Software has many dimensions along which we might look for modularity. Some of these dimensions might be:

a. program coding structure—blocks, statements, subroutines, etc.
b. program properties—name, priority, resource requirements, etc.
c. data structure—arrays, Cartesian products, records, files, libraries, etc.
d. data properties—function, formats, access conditions, modification conditions, etc.
e. physical representation—source files, object modules, load modules, overlay segments, headers, etc.
f. reference binding time—run, load, request, link, compile, library entry, system generation, etc.
g. physical location—in storage media with different access times.

The above list is not as complete or as sharply drawn as I would like, but the point is that there are multiple dimensions to software, and Unique Definition can be applied to all of them. The previous section looked at the design of IMPAC only in the dimension of data properties of a certain data structure, Block; other dimensions were not discussed. For example, the Operator "program" in Figure 4 is actually a system of program and data modules, generated from several types of source modules into several kinds of physical modules.

It is very interesting to note that the IMPAC design as described above exhibits the kind of modularity proposed by Parnas,[3] although the approach was slightly different. Instead of "information hiding,"[4] the criterion was "Unique Definition." Information hiding seems to be the same kind of concept as Unique Definition, but stated in a negative way; for if the design decisions in a program module are hidden from other program modules, then those design decisions must be uniquely defined in that module. Improvements in changeability, independent development, comprehensibility, and reliability are achieved with either approach. Parnas also predicted that this kind of modularity could require unusual implementation methods,[3] especially to achieve good efficiency, and this is also reflected in the FOX 1 programming system.

However, Unique Definition of software constructs seems (to me, at least) to be a broader concept than information hiding, since it applies to the entire programming process, not just program design. (Psychological note: I have met people who are quite reluctant to have information hidden from them, but who on the other hand are quite willing to reference unique definitions!)

APPLICATION NOTES

Applying the principle requires tools which facilitate defining, storing, and referencing Unique Definitions of software constructs. Some tools for these purposes were mentioned earlier. Programming systems can be graded by the tools they provide for Unique Definition,[6] and the ease of constructing new tools. As a general guideline, tools should present as few opportunities for error as possible, which means mainly to eliminate human involvement as much as possible.

Very often, referencing a unique physical definition at execution time is either impossible or too inefficient, so practical considerations force us to distribute copies and/or transformations of a unique definition into multiple physical locations. For example, distribution of unique IMPAC Internal and Record format definitions to Execute program and GLOB coding was done to improve efficiency, while distribution of a new release of standard software modules by a computer manufacturer to his customers is necessitated by the obvious impracticality of connecting all the computers to a common memory at the manufacturer's plant. But sooner or later it will be necessary to modify such a definition and reestablish all its physical forms, so a problem is to devise processes for doing this which offer the fewest opportunities for error. This involves:

a. knowing all the physical locations of a definition,
b. distributing the new definition,
c. rendering the definition into the appropriate physical form(s),
d. physically installing the new forms,
e. detecting any errors,
f. knowing when the process is complete.

Ideally, such a process should be fast and easy to use. If it is instead slow and cumbersome, a person may be forced to devise an alternate process which is faster but less secure, e.g., "patching," especially if he is under heavy schedule pressures.

Perhaps the most valuable tools are those which can find and list all the references to a definition. Examples:

- which statements reference a given variable; which potentially modify it,
- which programs INCLUDE a given file,
- which programs OPEN a given file,
- which job streams invoke a given utility program,
- which load modules contain a given library subroutine.

This problem can be quite severe on large, highly modular systems—but if we wish to practice modular design, we must face the problem of keeping track of all the modules. For IMPAC, this has meant maintaining a list of all modules, their types and connectivity.

Unique Definition can also be applied to software specification, design, design review, manufacturing, testing, maintenance, and (especially) documentation. Outside the area of software, Unique Definition, under other names, has long been used. For instance, custody of the unique definitions of physical measurements for the U.S.A. is the responsibility of the National Bureau of Standards; the process of referencing such a definition is called calibration.

CONCLUSION

Unique Definition of software constructs is proposed as a design approach to improve software reliability, by reducing the opportunities for error. It is a principle which can be applied to the entire programming process. The two continuing challenges in applying Unique Definition are recognizing software constructs and developing appropriate tools.

REFERENCES

1. Dahl, O. J., E. W. Dijkstra, C. A. R. Hoare, *Structured Programming,* Academic Press, 1972, London.
2. Griem, P. D., T. L. Willmott, *Programming Packed Process Data,* 1975 IFAC/IFIP Workshop on Real-Time Programming, Boston, Mass.
3. Parnas, D. L., "On the Criteria to be Used in Decomposing Systems into Modules," *Comm. ACM* 15, 12, December 1972, pp. 1053-1058.
4. Parnas, D. L., "Information Distribution Aspects of Design Methodology," *IFIP Congress 1971,* Ljubljana, Yugoslavia.
5. Griem, P. D., "A Reentrant Programming Model and Implementation Methods," *1974 IFAC/IFIP Workshop on Real-Time Programming,* Budapest.
6. For example, a novel facility for uniquely defining new data types (e.g., complex numbers) and operations upon them is provided in: PEARL—A proposal for a process and experiment automation real-time language KFK-PDV1, Gesellschaft für Kernforschung MBH, Karlsruhe, FRG, April 1973.
7. Sammet, J. E., *Programming Languages: History and Fundamentals,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1969, P. 526.
8. Willmott, T. L., "A Survey of Software for Direct Digital Control," 1968 Instrument Society of America Annual Conference, New York City. Reprinted in *Minicomputers: Hardware, Software, and Applications,* edited by J. D. Schoeffler and R. H. Temple, IEEE Press, 1972.
9. Waite, W. M., "The Mobile Programming System: STAGE 2," *Comm. ACM* 13, 7, July 1970, pp. 415-421.
10. Brown, P. J., "The ML/I Macro Processor," *Comm. ACM* 10, 10, October 1967, pp. 618-623.

APPENDIX A—GLOB CONTENTS

Typical information which a GLOB may contain is given below. A GLOB may also contain Assembly code when necessary to define special External formats or Rules for change.

General
Field name (6 characters)
Block types which have this field

C2. *Record format*
Starting word number
Starting bit number
Length in bits
Signed/unsigned
Left/right justified

C3. *Internal format*
INTEGER
REAL
DOUBLE PRECISION
character string (packed INTEGER array)
bit string (packed LOGICAL array)
Cartesian product

C4. *External format*
Fortran A, E, F, or I format
Octal
Binary
Block name
Process I/O addresses
Index: one of a set of character strings, to which the Internal format value is an index, e.g., 0/1/2/3=>OFF/LOW/OK/HI

C5. *Rules for change*
Changeable by
- Execute program
- Operator
- Engineer
- Application programs
- Generator

Range of acceptable values, where range limits are:
- the capacity of the Record format
- contained in the GLOB
- values in other fields of the Block record

Acceptable change from previous value
Change is conditional on the state of the Block as indicated by the values of other fields in the Block record.
Change is conditional on the state of the IMPAC system.

PDL—A tool for software design

by STEPHEN H. CAINE and E. KENT GORDON
Caine, Farber & Gordon, Inc.
Pasadena, California

INTRODUCTION

During the past several years, industry has seen an explosion in the cost of software production coupled with a decline in the quality and reliability of the results. A realization that structured programming, top-down design, and other changes in techniques can help has alerted the field to the importance of applying advanced design and programming methods to software production.[1,2]

For the past four years, Caine, Farber & Gordon, Inc. has used such advanced techniques as structured programming, top-down design and system implementation, centralized program production libraries, and egoless programming teams for all of its programming.[3-8] With these techniques we have achieved a level of productivity comparable to that recently reported by others employing similar techniques.

However, within the last year, we greatly refined these techniques, applying them to design as well as to programming. This has resulted in increased productivity, greatly decreased debugging effort, and clearly superior products. On recent complex projects we have achieved production rates, over the full development cycle, of 60-65 lines of finished code per man-day and computer utilization of less than 0.25 CPU hours per thousand lines of finished code. For comparison, these production rates are approximately half again better than our best efforts using just structured programming techniques and 4-6 times better than average industrial experience using classical techniques. Computer usage was four times smaller than our experience with just structured programming techniques and more than 10 times smaller than classical industrial averages.

As an example, consider the two CFG projects shown in Table 1. Project "A" is a major component of a seismic data processing system for oil exploration. It was produced using "classical" structured programming techniques and production rates compare favorably to other projects[3] which used similar techniques. Project "B" is a system for the automatic restructuring of Fortran programs.[9] It was developed using the latest CFG methods. Production rates were 50 percent better than for project "A" and the amount of computer time used in development was approximately one quarter of that used for the first project. In each case, a "line" of code was taken to be one 80-column source card with common data definitions counted only once. Both projects were developed using an IBM 370/158.

In order to achieve the results that we are currently experiencing, we have developed a comprehensive software production methodology which places its greatest emphasis on design. Before *any* code is written, a complete design is produced which contains:

- all external and internal interface definitions
- definitions of all error situations
- identification of all procedures
- identification of all procedure calls
- definition of all global data
- definition of all control blocks
- specification of the processing algorithms of all procedures

The design is produced and presented top-down and is oriented toward understandability by people. While in no sense is our design process automated, it is supported by a series of tools—both computerized and procedural.

This paper is not intended to present our complete design and implementation methodology. Rather, it discusses one of the design tools—the "Program Design Language" (PDL) and its computerized processor. Both of these have been in extensive use since the autumn of 1973.

THE PURPOSE OF PDL

PDL is designed for the production of structured designs in a top-down manner. It is a "pidgin" language in that it uses the vocabulary of one language (i.e., English) and the overall syntax of another (i.e., a structured programming language). In a sense, it can be thought of as "structured English."

While the use of pidgin languages is also advocated by others, we have taken the additional steps of imposing a degree of formalism on the language and supplying a processor for it. Input to the processor consists of control information plus designs for procedures (called "segments" in PDL). The output is a working design document which can, if desired, be photo-reduced and included in a project development workbook.

The output of the processor completely replaces flowcharts since PDL designs are easier to produce, easier to change, and easier to read than are designs presented in flowchart form.

TABLE 1—Production Comparisons

	PROJECT "A"	PROJECT "B"
DEVELOPMENT METHOD	CLASSICAL STRUCTURED	LATEST CFG
PROGRAMMING LANGUAGE	PL/I DIALECT	PL/I
SIZE OF PROGRAM (LINES)	32,000	27,000
SIZE OF TEAM	3–6	3–5
ELAPSED TIME (MONTHS)	9	6
LINES PER MAN-DAY	40	65
CPU HOURS PER 1000 LINES (IBM 370/158)	0.90	0.16

DESIGNING FOR PEOPLE IN PDL

Like a flowchart, and unlike a program, PDL can be written with whatever level of detail is appropriate to the problem at hand. A designer can start with a few pages giving the general structure of his system and finish, if necessary, with even more precision than would exist in the corresponding program.

In our experience, the purpose of a design is to communicate the designer's idea to other people—not to a computer. Figure 1 shows a sample design "segment" for a simple exchange sort. Note that we are *not* attempting to illustrate efficient sorting methods. Rather, having decided to use this particular sorting method, we wish to present the algorithm in a way that it can be easily comprehended. Given that the "DO UNTIL" construct represents a loop whose completion test occurs at the *end* of the loop, the operation of the algorithm is apparent. It is clearly better, from the viewpoint of understandability, than either the flowchart of Figure 2 or the translation of the algorithm into PL/I as shown in Figure 3.

A virtue of PDL is that a rough outline of an entire problem solution can be quickly constructed. This level of design can be easily understood by people other than the designer. Thus, criticisms, suggestions, and modifications can be quickly incorporated into the design, possibly resulting in complete rewrites of major sections. When the design has stabilized at this level, more detail can be added in successive passes through the design with decisions at each point affecting smaller and smaller areas.

```
SORT (TABLE, SIZE OF TABLE)

 IF SIZE OF TABLE > 1
   DO UNTIL NO ITEMS WERE INTERCHANGED
     DO FOR EACH PAIR OF ITEMS IN TABLE (1-2, 2-
           3, 3-4, ETC.)
       IF FIRST ITEM OF PAIR > SECOND ITEM OF
           PAIR
         INTERCHANGE THE TWO ITEMS
       ENDIF
     ENDDO
   ENDDO
 ENDIF
```

Figure 1—PDL design of a simple sorting algorithm

THE FORM OF A DESIGN IN PDL

A design produced in PDL consists of a number of "flow segments," each corresponding roughly to a procedure in the final implementation. A sample of a high-level flow segment from a large design is shown in Figure 4. If a statement in a segment references another flow segment,

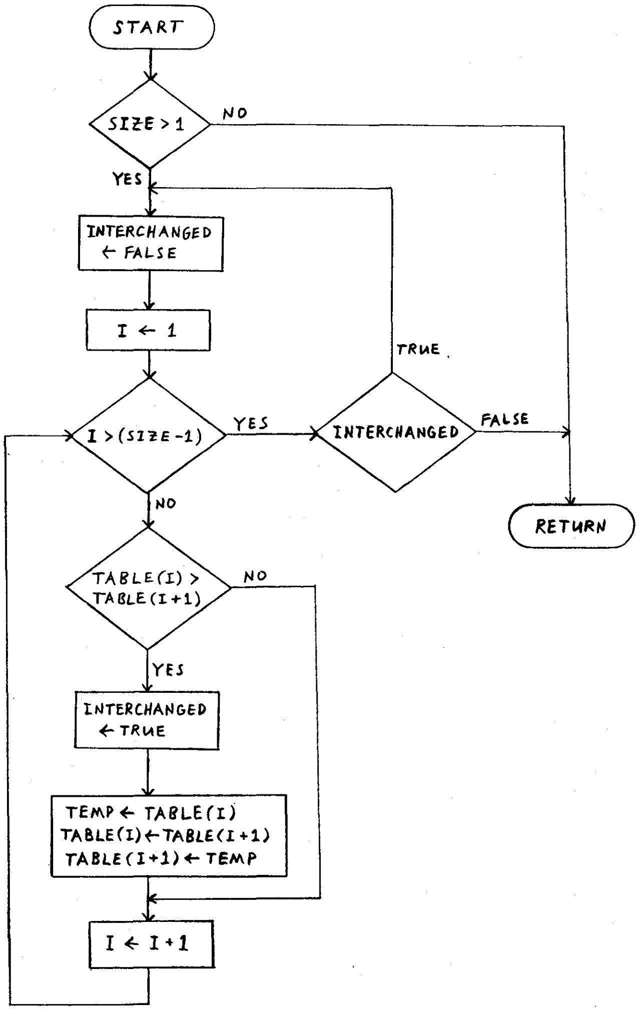


Figure 2—Flowchart for sorting algorithm of Figure 1

```
SORT:
  PROCEDURE(TABLE);
  DECLARE TABLE(*) FIXED BIN;
  DECLARE INTERCHANGED BIT(1);
  DECLARE TEMP FIXED BIN;
  IF DIM(TABLE,1) > 1 THEN
    DO;
      INTERCHANGED='1'B;
      DO WHILE (INTERCHANGED);
        INTERCHANGED='0'B;
        DO I=LBOUND(TABLE,1) TO
        HBOUND(TABLE,1)-1;
          IF TABLE(I)>TABLE(I+1) THEN
            DO;
              INTERCHANGED='1'B;
              TEMP=TABLE(I);
              TABLE(I)=TABLE(I+1);
              TABLE(I+1)=TEMP;
            END;
          END;
        END;
      END;
 END SORT;
```

Figure 3—PL/I procedure for sorting algorithm

the page number of the referenced segment is shown to the left of the referencing statement. A sample low-level segment is shown in Figure 5.

The statements which compose a flow segment are entered in free form. The PDL processor automatically underlines keywords, indents statements to correspond to structure nesting levels, and provides automatic continuation from line to line.

Design information may also be entered in "text segments." These contain purely textual information such as commentary, data formats, assumptions, and constraints.

The document output by the PDL processor is in a form ready for photo-reduction and publication. It contains:

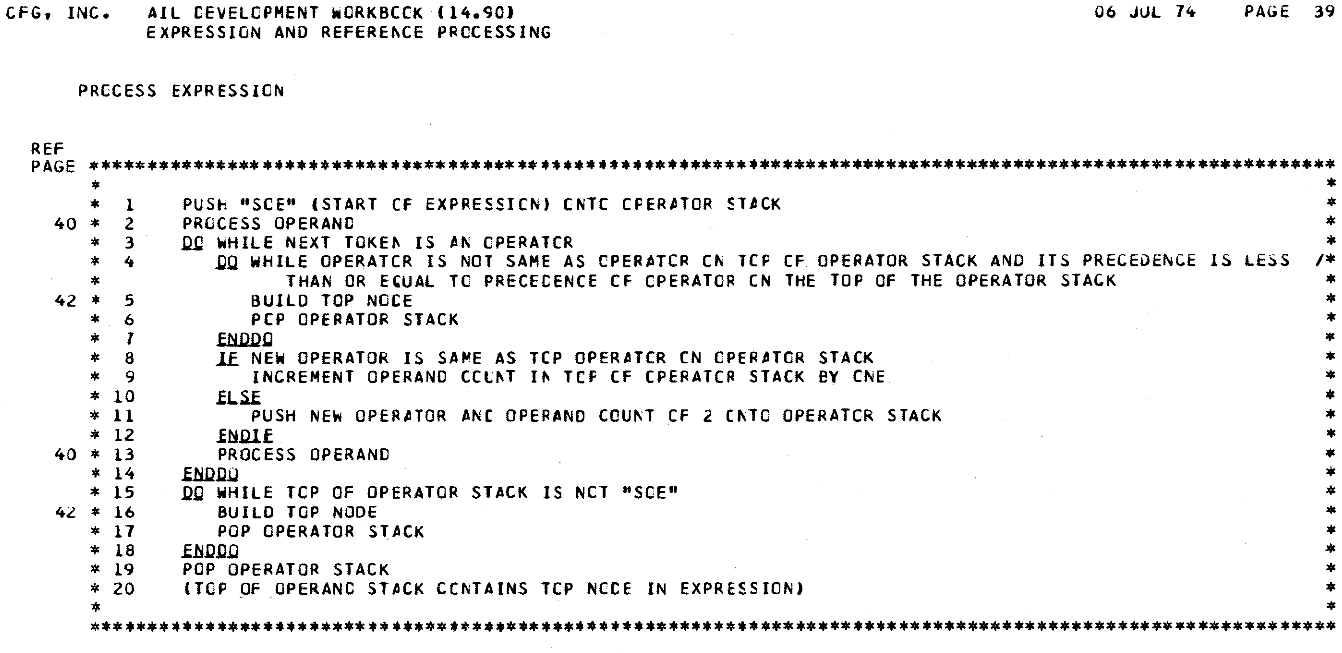
- a cover page giving the design title, data, and processor identification
- a table of contents (Figure 6)
- the body of the design, consisting of flow segments and text segments
- a "reference tree" showing how segment references are nested (Figure 7)
- a cross-reference listing showing the page and line number at which each segment is referenced (Figure 8)

DESIGN CONSTRUCTS

What goes into a design segment is generally at the discretion of the designer. In choosing the form of presentation, he is guided by a compendium of style which has been developed through extensive experience. However, the language and the processor have been defined to encourage and support design constructs which relate directly to the constructs of structured coding. The two primary constructs are the IF and the DO.

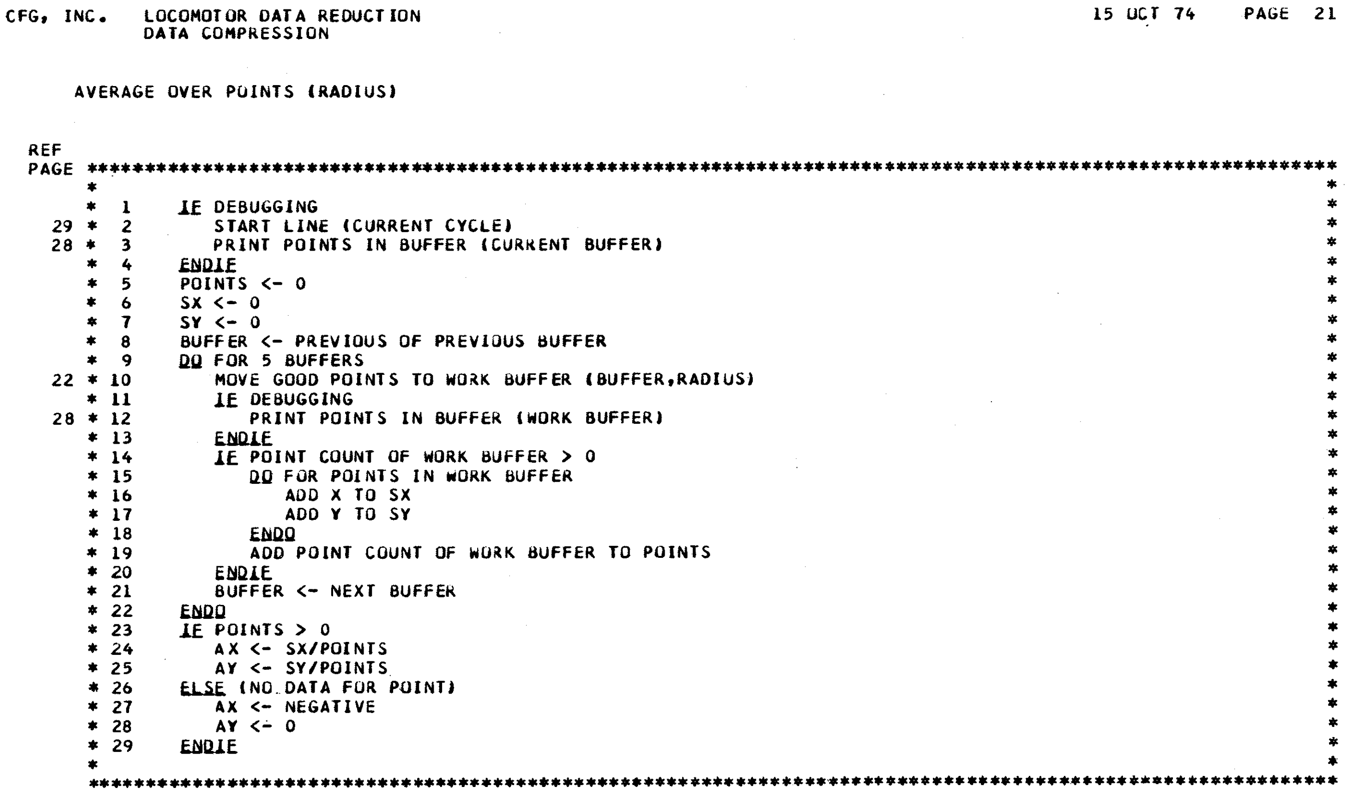
The IF construct

The IF construct provides the means for indicating conditional execution. It corresponds to the classical IF . . . THEN . . . ELSE construct of Algol-60[10] and PL/I,



```
CFG, INC.   AIL CEVELCPMENT WORKBCCK (14.90)                                                    06 JUL 74     PAGE  39
            EXPRESSION AND REFERENCE PROCESSING


     PRCCESS EXPRESSION


  REF
  PAGE *************************************************************************************************************
       *                                                                                                            *
       *  1    PUSH "SCE" (START CF EXPRESSICN) CNTC CPERATOR STACK                                                 *
    40 *  2    PRCCESS OPERANC                                                                                      *
       *  3    DC WHILE NEXT TOKEN IS AN CPERATCR                                                                   *
       *  4       DO WHILE OPERATCR IS NOT SAME AS CPERATCR CN TCP CF OPERATOR STACK AND ITS PRECEDENCE IS LESS    /*
       *                THAN OR ECUAL TC PRECECENCE CF CPERATOR CN THE TOP OF THE OPERATOR STACK                    *
    42 *  5          BUILD TOP NCCE                                                                                 *
       *  6          PCP OPERATOR STACK                                                                             *
       *  7       ENDDO                                                                                             *
       *  8       IF NEW OPERATOR IS SAME AS TCP OPERATCR CN CPERATCR STACK                                         *
       *  9          INCREMENT OPERAND CCUNT IN TCF CF CPERATCR STACK BY CNE                                        *
       * 10       ELSE                                                                                              *
       * 11          PUSH NEW OPERATOR ANC OPERAND COUNT CF 2 CNTC OPERATCR STACK                                   *
       * 12       ENDIE                                                                                             *
    40 * 13       PROCESS OPERAND                                                                                   *
       * 14    ENDDO                                                                                                *
       * 15    DO WHILE TCP OF OPERATOR STACK IS NCT "SCE"                                                          *
    42 * 16       BUILD TOP NODE                                                                                    *
       * 17       POP OPERATOR STACK                                                                                *
       * 18    ENDDO                                                                                                *
       * 19    PCP OPERATOR STACK                                                                                   *
       * 20    (TCP OF OPERANC STACK CCNTAINS TCP NCCE IN EXPRESSION)                                               *
       *                                                                                                            *
       **************************************************************************************************************
```

Figure 4—Sample of a high-level PDL flow segment



```
CFG, INC.   LOCOMOTOR DATA REDUCTION                                                    15 OCT 74     PAGE  21
            DATA COMPRESSION

     AVERAGE OVER POINTS (RADIUS)

 REF
 PAGE ***********************************************************************************************************
      *                                                                                                         *
      *  1     IF DEBUGGING                                                                                     *
   29 *  2        START LINE (CURRENT CYCLE)                                                                    *
   28 *  3        PRINT POINTS IN BUFFER (CURRENT BUFFER)                                                       *
      *  4     ENDIF                                                                                            *
      *  5     POINTS <- 0                                                                                      *
      *  6     SX <- 0                                                                                          *
      *  7     SY <- 0                                                                                          *
      *  8     BUFFER <- PREVIOUS OF PREVIOUS BUFFER                                                            *
      *  9     DO FOR 5 BUFFERS                                                                                 *
   22 * 10        MOVE GOOD POINTS TO WORK BUFFER (BUFFER,RADIUS)                                               *
      * 11        IF DEBUGGING                                                                                  *
   28 * 12           PRINT POINTS IN BUFFER (WORK BUFFER)                                                       *
      * 13        ENDIF                                                                                         *
      * 14        IF POINT COUNT OF WORK BUFFER > 0                                                             *
      * 15           DO FOR POINTS IN WORK BUFFER                                                               *
      * 16              ADD X TO SX                                                                             *
      * 17              ADD Y TO SY                                                                             *
      * 18           ENDDO                                                                                      *
      * 19           ADD POINT COUNT OF WORK BUFFER TO POINTS                                                   *
      * 20        ENDIF                                                                                         *
      * 21        BUFFER <- NEXT BUFFER                                                                         *
      * 22     ENDDO                                                                                            *
      * 23     IF POINTS > 0                                                                                    *
      * 24        AX <- SX/POINTS                                                                               *
      * 25        AY <- SY/POINTS                                                                               *
      * 26     ELSE (NO DATA FOR POINT)                                                                         *
      * 27        AX <- NEGATIVE                                                                                *
      * 28        AY <- 0                                                                                       *
      * 29     ENDIF                                                                                            *
      *                                                                                                         *
      ***********************************************************************************************************
```

Figure 5—Sample low-level PDL flow segment

augmented by the ELSEIF of languages such as Algol-68.[11] The latter is used to prevent excessive indentation levels when cascaded tests are used.

The general form of the construct is shown in Figure 9. Any number (including zero) ELSEIF's are allowed and at most one ELSE is allowed.

The DO construct

This construct is used to indicate repeated execution and for case selection. The reasons for the dual use of this construct are historic in nature and closely map several of the in-house implementation languages we frequently use. It may be effectively argued that a separate construct for case selection would be better.

The iterative DO is indicated by:

```
DO iteration criteria
   one or more statements
ENDDO
```

The "iteration criteria" can be chosen to suit the problem. As always, bias toward human understandability is preferred. Statements such as:

DO WHILE THERE ARE INPUT RECORDS
DO UNTIL "END" STATEMENT HAS BEEN
PROCESSED
DO FOR EACH ITEM IN THE LIST EXCEPT
THE LAST ONE

occur frequently in actual designs.

Our experience, and that of others,[7] has shown that a provision for premature exit from a loop and premature repetition of a loop are frequently useful. To accomplish this, we take the statement

UNDO

to mean that control is to pass to the point following the ENDDO of the loop. Likewise,

CYCLE

is taken to mean that control is to pass to the loop termination test.

Since we may wish that an UNDO or CYCLE apply to an outer loop in a nest of loops, any DO may be labelled and the label may be placed after the UNDO or CYCLE.

Case selection is indicated by

DO CASE selection criteria

Again, we advocate the use of understandable selection criteria such as

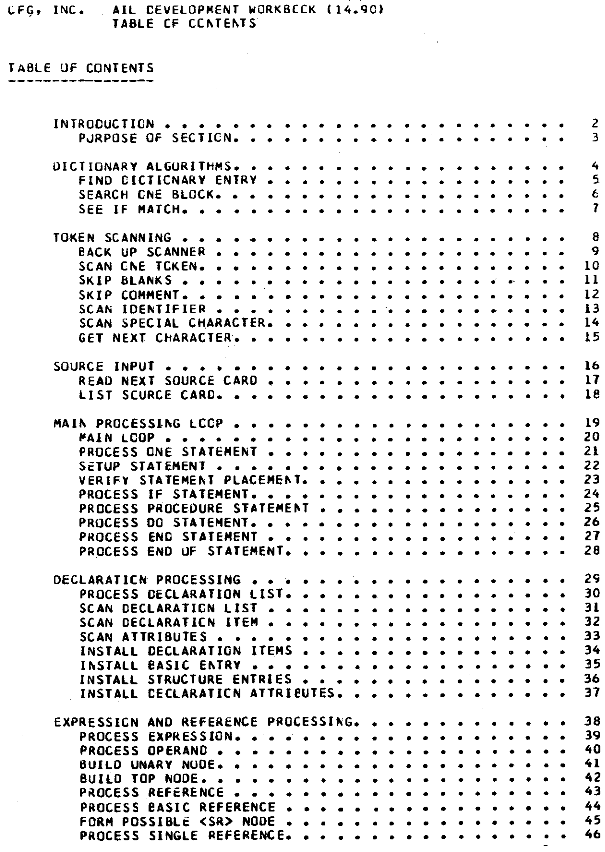
DO CASE OF TRANSACTION TYPE
DO CASE OPERATOR TYPE
DO CASE OF CONTROL CARD VERB

Generally, we use labels in the body of the DO to indicate where control passes for each case. This is illustrated in Figure 10.

FUTURE DIRECTIONS

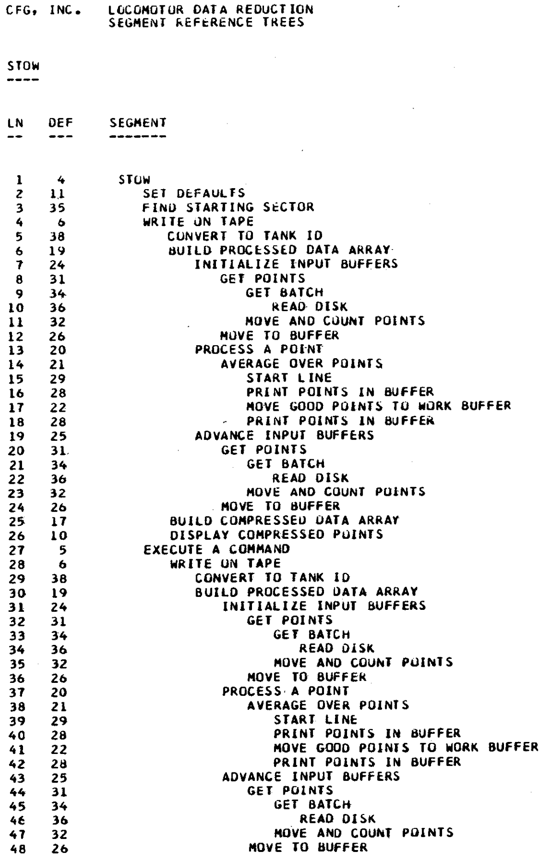
The results we have achieved with PDL have exceeded our original expectations. However, it is clear that further development is both possible and desirable. The areas which we are currently exploring include:

- *handling of data*: The current PDL presents a procedural design—a design of control flow and processing actions. It would be very desirable to have a similar mechanism for the design of data structures and data flow. A method for integrating the data and procedural designs and performing mutual cross-referencing would be very powerful, indeed.
- *interactive versions*: the current PDL processor is batch oriented. The ability to compose and, more importantly, to modify a design on-line in a manner specifically planned for interactive use would be of great assistance. This would be particularly advantageous during the early stages of a project when design changes are often frequent and extensive.



```
CFG, INC.   AIL CEVELOPMENT WORKBCCK (14.90)
            TABLE CF CCNTENTS


TABLE UF CONTENTS
-----------------


      INTRODUCTION . . . . . . . . . . . . . . . . . . . . . . . .   2
         PJRPOSE OF SECTICN. . . . . . . . . . . . . . . . . . . .   3

      DICTIONARY ALGORITHMS. . . . . . . . . . . . . . . . . . . .   4
         FIND CICTICNARY ENTRY . . . . . . . . . . . . . . . . . .   5
         SEARCH CNE BLOCK. . . . . . . . . . . . . . . . . . . . .   6
         SEE IF MATCH. . . . . . . . . . . . . . . . . . . . . . .   7

      TOKEN SCANNING . . . . . . . . . . . . . . . . . . . . . . .   8
         BACK UP SCANNER . . . . . . . . . . . . . . . . . . . . .   9
         SCAN CNE TCKEN. . . . . . . . . . . . . . . . . . . . . .  10
         SKIP BLANKS . . . . . . . . . . . . . . . . . . . . . . .  11
         SKIP COMMENT. . . . . . . . . . . . . . . . . . . . . . .  12
         SCAN IDENTIFIER . . . . . . . . . . . . . . . . . . . . .  13
         SCAN SPECIAL CHARACTER. . . . . . . . . . . . . . . . . .  14
         GET NEXT CHARACTER. . . . . . . . . . . . . . . . . . . .  15

      SOURCE INPUT . . . . . . . . . . . . . . . . . . . . . . . .  16
         READ NEXT SOURCE CARD . . . . . . . . . . . . . . . . . .  17
         LIST SCURCE CARD. . . . . . . . . . . . . . . . . . . . .  18

      MAIN PROCESSING LCCP . . . . . . . . . . . . . . . . . . . .  19
         MAIN LCOP . . . . . . . . . . . . . . . . . . . . . . . .  20
         PROCESS ONE STATEMENT . . . . . . . . . . . . . . . . . .  21
         SETUP STATEMENT . . . . . . . . . . . . . . . . . . . . .  22
         VERIFY STATEMENT PLACEMENT. . . . . . . . . . . . . . . .  23
         PROCESS IF STATEMENT. . . . . . . . . . . . . . . . . . .  24
         PROCESS PROCEDURE STATEMENT . . . . . . . . . . . . . . .  25
         PROCESS DO STATEMENT. . . . . . . . . . . . . . . . . . .  26
         PROCESS ENC STATEMENT . . . . . . . . . . . . . . . . . .  27
         PROCESS END UF STATEMENT. . . . . . . . . . . . . . . . .  28

      DECLARATICN PROCESSING . . . . . . . . . . . . . . . . . . .  29
         PROCESS DECLARATION LIST. . . . . . . . . . . . . . . . .  30
         SCAN DECLARATICN LIST . . . . . . . . . . . . . . . . . .  31
         SCAN DECLARATICN ITEM . . . . . . . . . . . . . . . . . .  32
         SCAN ATTRIBUTES . . . . . . . . . . . . . . . . . . . . .  33
         INSTALL DECLARATION ITEMS . . . . . . . . . . . . . . . .  34
         INSTALL BASIC ENTRY . . . . . . . . . . . . . . . . . . .  35
         INSTALL STRUCTURE ENTRIES . . . . . . . . . . . . . . . .  36
         INSTALL CECLARATICN ATTRIBUTES. . . . . . . . . . . . . .  37

      EXPRESSICN AND REFERENCE PROCESSING. . . . . . . . . . . . .  38
         PROCESS EXPRESSION. . . . . . . . . . . . . . . . . . . .  39
         PROCESS OPERAND . . . . . . . . . . . . . . . . . . . . .  40
         BUILD UNARY NODE. . . . . . . . . . . . . . . . . . . . .  41
         BUILD TOP NODE. . . . . . . . . . . . . . . . . . . . . .  42
         PROCESS REFERENCE . . . . . . . . . . . . . . . . . . . .  43
         PROCESS BASIC REFERENCE . . . . . . . . . . . . . . . . .  44
         FORM POSSIBLE <SR> NODE . . . . . . . . . . . . . . . . .  45
         PROCESS SINGLE REFERENCE. . . . . . . . . . . . . . . . .  46
```

Figure 6—Sample table of contents from a PDL design



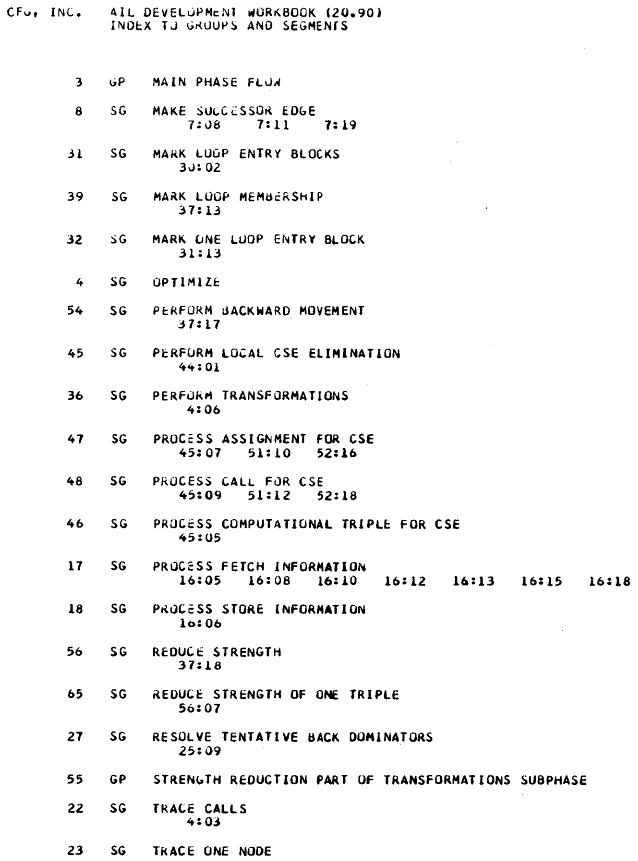
```
CFG, INC.   LOCOMOTOR DATA REDUCTION
            SEGMENT REFERENCE TREES


STOW
----


LN   DEF    SEGMENT
--   ---    -------


 1    4      STOW
 2   11         SET DEFAULTS
 3   35         FIND STARTING SECTOR
 4    6         WRITE ON TAPE
 5   38            CONVERT TO TANK ID
 6   19            BUILD PROCESSED DATA ARRAY
 7   24               INITIALIZE INPUT BUFFERS
 8   31                  GET POINTS
 9   34                     GET BATCH
10   36                        READ DISK
11   32                     MOVE AND COUNT POINTS
12   26                  MOVE TO BUFFER
13   20               PROCESS A POINT
14   21                  AVERAGE OVER POINTS
15   29                     START LINE
16   28                     PRINT POINTS IN BUFFER
17   22                     MOVE GOOD POINTS TO WORK BUFFER
18   28                     PRINT POINTS IN BUFFER
19   25               ADVANCE INPUT BUFFERS
20   31                  GET POINTS
21   34                     GET BATCH
22   36                        READ DISK
23   32                     MOVE AND COUNT POINTS
24   26                  MOVE TO BUFFER
25   17            BUILD COMPRESSED DATA ARRAY
26   10            DISPLAY COMPRESSED POINTS
27    5         EXECUTE A COMMAND
28    6            WRITE ON TAPE
29   38               CONVERT TO TANK ID
30   19               BUILD PROCESSED DATA ARRAY
31   24                  INITIALIZE INPUT BUFFERS
32   31                     GET POINTS
33   34                        GET BATCH
34   36                           READ DISK
35   32                        MOVE AND COUNT POINTS
36   26                     MOVE TO BUFFER
37   20                  PROCESS A POINT
38   21                     AVERAGE OVER POINTS
39   29                        START LINE
40   28                        PRINT POINTS IN BUFFER
41   22                        MOVE GOOD POINTS TO WORK BUFFER
42   28                        PRINT POINTS IN BUFFER
43   25                  ADVANCE INPUT BUFFERS
44   31                     GET POINTS
45   34                        GET BATCH
46   36                           READ DISK
47   32                        MOVE AND COUNT POINTS
48   26                     MOVE TO BUFFER
```

Figure 7—Sample of a segment reference tree

- *total design system*: an integrated computer system for software design, such as the DES system of Professor R. M. Graham,[12] is a natural outgrowth of our work with PDL. Such a system would act as an information management system maintaining a data base of designs. Designs could be entered and modified; questions about a design and the inter-relations of its parts could be asked and answered; reports on design status and completeness could be prepared. Provision for simulation of a design for performance estimation and a mechanism for transition from design to code are also important.

CONCLUSIONS

In the autumn of 1973, we integrated the use of PDL and its processor into our software design and implementation methodology. Since then, it has been used on a number of



```
CFG, INC.    AIL DEVELOPMENT WORKBOOK (20.90)
             INDEX TO GROUPS AND SEGMENTS


      3   GP   MAIN PHASE FLOW

      8   SG   MAKE SUCCESSOR EDGE
                  7:08     7:11     7:19

     31   SG   MARK LOOP ENTRY BLOCKS
                 30:02

     39   SG   MARK LOOP MEMBERSHIP
                 37:13

     32   SG   MARK ONE LOOP ENTRY BLOCK
                 31:13

      4   SG   OPTIMIZE

     54   SG   PERFORM BACKWARD MOVEMENT
                 37:17

     45   SG   PERFORM LOCAL CSE ELIMINATION
                 44:01

     36   SG   PERFORM TRANSFORMATIONS
                  4:06

     47   SG   PROCESS ASSIGNMENT FOR CSE
                 45:07    51:10    52:16

     48   SG   PROCESS CALL FOR CSE
                 45:09    51:12    52:18

     46   SG   PROCESS COMPUTATIONAL TRIPLE FOR CSE
                 45:05

     17   SG   PROCESS FETCH INFORMATION
                 16:05    16:08    16:10    16:12    16:13    16:15    16:18

     18   SG   PROCESS STORE INFORMATION
                 16:06

     56   SG   REDUCE STRENGTH
                 37:18

     65   SG   REDUCE STRENGTH OF ONE TRIPLE
                 56:07

     27   SG   RESOLVE TENTATIVE BACK DOMINATORS
                 25:09

     55   GP   STRENGTH REDUCTION PART OF TRANSFORMATIONS SUBPHASE

     22   SG   TRACE CALLS
                  4:03

     23   SG   TRACE ONE NODE
```

Figure 8—Part of an index to a design

projects of varying sizes. The results have been comparable to those discussed earlier.

PDL is not a "panacea" and it is certainly possible to produce bad designs using it. However, we have found that our designers and programmers quickly learn to use PDL effectively. Its emphasis on designing for people provides a high degree of confidence in the correctness of the design. In our experience, it is almost impossible to "wave your hands" in PDL. If a designer doesn't really yet see how to solve a particular problem, he can't just gloss over it without the resulting design gap being readily apparent to a reader of the design. This, plus the basic readability of a PDL design, means that clients, management, and team members can both understand the proposed solution and gauge its degree of completeness.

We have also found that PDL works equally well for large and small projects. Because it is so easy to use, persons starting to work on even a "quick and dirty" utility will first sketch out a solution in PDL. In the past, such programs were usually written with little or no design preceding the actual coding.

```
IF condition
  one or more statements
ELSEIF condition
  one or more statements
  .
  .
  .
ELSEIF condition
  one or more statements
ELSE
  one or more statements
ENDIF
```

Figure 9—General form of IF construct

```
   DO CASE OF TRANSACTION TYPE
ADD:
      CREATE INITIAL RECORD
DELETE:
      IF DELETION IS AUTHORIZED
        CREATE DELETION RECORD
      ELSE
        ISSUE ERROR MESSAGE
      ENDIF
CHANGE:
      INCREMENT CHANGE COUNT
      CREATE DELETION RECORD
"OTHER":
      ISSUE ERROR MESSAGE
   ENDO
```

Figure 10—Example of DO CASE construct

REFERENCES

1. Boehm, B. W., "Software and its Impact: A Quantitative Assessment," *Datamation,* May 1973, pp. 48-59.
2. Goldberg, J., (editor), *Proceedings of a Symposium on the High Cost of Software,* Stanford Research Institute, 1973.
3. Baker, F. T., "Chief Programmer Team Management of Production Programming," *IBM Sys. J.,* Vol. 11, No. 1, 1972, pp. 56-73.
4. Bohm, C. and G. Jacopini, "Flow Diagrams, Turing Machines and Languages With Only Two Formation Rules," *Comm. ACM,* May 1966, pp. 366-371.
5. Dijkstra, E., "GO TO Statements Considered Harmful," *Comm. ACM,* March 1968, pp. 147-148.
6. Mills, Harlan D., "On the Development of Large Reliable Programs," *IEEE Symp. Computer Software Reliability,* 1973, pp. 155-159.
7. Peterson, W. W., T. Kasami and N. Tokura, "On the Capabilities of WHILE, REPEAT and EXIT Statements," *Comm. ACM,* August 1973, pp. 503-512.
8. Stevens, W. P., G. J. Myers and L. L. Constantine, "Structured Design," *IBM Sys. J.,* Vol. 13, No. 2, 1974, pp. 115-139.
9. De Balbine, G., *Better Manpower Utilization Using Automatic Restructuring* Caine, Farber & Gordon, Inc., 1974 (in publication).
10. Naur, P. et al., "Report on the Algorithmic Language ALGOL 60," *Comm. ACM,* May 1960, pp. 299-314.
11. Van Wijngaarden, A. et al., "Report on the Algorithmic Language ALGOL 68," *Numerische Mathematik,* 14, 1969, pp. 79-218.
12. Graham, R. M., G. J. Clancy and D. B. Devaney, "A Software Design and Evaluation System," *Comm. ACM,* February 1973, pp. 110-116.

Structured programming and structured design as art forms

by EDWARD YOURDON
Yourdon Inc.
New York, New York

It is generally known that structured programming was initially promoted in Europe through the efforts of Edsger Dijkstra and others. However, its introduction to the USA is primarily due to the efforts of IBM; furthermore, much of the American programmer's knowledge of structured programming is based on such superficial discussions as the December 1973 *Datamation* articles.

The consequences of this superficial exposure are beginning to be realized: programming without the GO-TO still offers innumerable ways of writing incomprehensible programs. More important, it is now being realized that structured programming imposes a rigid discipline on the *coding* of a program, but leaves important *design* questions to the whim of the programmer. Thus, we see a number of "structured" programs whose modules share local working storage with one another, or whose code manages to ALTER the code in other modules; the result is that an innocent change to one module causes unpredictable problems in another module.

Because of these problems, there is growing interest in a related discipline known as "structured design." Based on work carried out by Larry Constantine in 1965-68, structured design concentrates on the relationships between modules; it introduces the concepts of "coupling" and "cohesiveness" to help quantify the "goodness" of a design. When carried out properly, a program implemented with structured design is substantially easier to maintain than a program implemented with either a "random-design" approach or even a "structured programming" approach. This is particularly interesting since structured design does not impose any rigid disciplines on the code within a module. Thus, structured design combined with structured programming should be an almost unbeatable combination.

Unfortunately, structured design is not yet a precise science. For example, coupling is easily understood by programmers; however the strategy of minimizing intermodule coupling often contradicts the programmer's instincts and habits: for reasons of efficiency or convenience, he is often unwilling to relinquish his use of global variables, COMMON data areas, and common control blocks. Cohesiveness is similarly misunderstood: the programmer learns that a module called EDIT-AND-UPDATE has "sequential" cohesiveness, and that EDIT-ALL-TRANSACTIONS has "logical" cohesiveness—and that neither module is considered "good" when compared to the ideal module possessing "functional" cohesiveness. Since the programmer often doesn't understand the various levels of cohesiveness anyway, and since it may contradict design strategies he has employed for years, he may ignore the suggestions of structured design altogether.

To summarize, programming *has* improved since the introduction of structured programming and structured design. However, it is unreasonable to think that programming has suddenly become a rigorous science—and it is foolish to think that the application of structured programming is *guaranteed* to lead to "perfect" computer systems. Our programmers still need more education in the "art" of structured programming; they need more experience in the application of structured programming and structured design to real-world systems; and finally, they need "rules" of structured design that are as rigid as the "no-GO-TO" rule is in the realm of coding.

Modularization around a suitable abstraction

by STEPHEN N. ZILLES
IBM Research
San Jose, California

Suitable modularization is a good programming practice. It permits parallel development of modules, component testing as the pieces are completed, and simpler maintenance. Achieving these properties depends on defining clean interfaces and minimizing the dependencies between modules.

Most methodologies for decomposing a program into modules place too much emphasis on control flow. Too little attention is given to how data usage should affect the choice of modules. Such conventions as COBOL's separate data division (putting all working data at the top level of the module hierarchy, and passing all information in the parameter list) give modules access to data which they do not need, but which they might use to improve local efficiency. This use of unneeded data produces dependencies among modules which logically should be unrelated.

The data which might produce unwanted dependencies is not just data unrelated to the function of the module. The primary problem occurs with data items which describe or are part of the representation of the data structures to be processed by the module. As a simple example, consider the implementation of a relational data base. The data items being related are important to the processing programs. The data items used in representing the relations—field sizes, pointers—should not be used in the processing module because that module should be insensitive to representational changes.

By choosing a suitable abstraction of the data we can present only the functionally necessary aspects of the data representation and suppress implementation details. Procedural abstraction has been used to modularize a sequence of simple actions and construct a higher level action with a simpler interface, but little use has been made of abstractions built around data. A data abstraction consists of a set of abstract objects—a data base; a print spooler; models of machines in a machine shop—and the operations which can be performed on the objects. The operations manipulate the visible aspects of the abstraction—the items in the data base, print files, the parts of the machine—and suppress access to the implementation details—the representation of the relations, the number of printers, the representation of the parts. This allows some aspects of the data abstraction to be computed rather than represented directly, and, conversely, allows aspects to be redundantly represented and insures the consistency of the representation.

Since a data abstraction includes both data objects and the operations applicable to them, a suitable modularization must include procedures for each of the operations. This multiprocedure module encapsulates the representational information and makes it inaccessible outside the module. Hence, dependencies are kept within the module as desired. Because most existing languages do not support the construction of such modules, the ideal solution would be to design new languages which do. Practical considerations, however, require that we also develop ways to use existing languages to implement multi-procedure modules. This can be done using preprocessors and management control techniques. The important thing is to recognize and consider data abstractions when the modularization is chosen, and thereby limit data access to relevant operations for manipulating it.

Minicompilers, preprocessors and other tools

by P. J. PLAUGER
Bell Laboratories
Murray Hill, New Jersey

Building on the work of others is the only way to make substantial progress in any field. Yet computer programming continues as a cottage industry because programmers insist on reinventing tools for each new application. What we must encourage is a way of packaging programs so that they can be *perceived* as standard tools, each performing its specialized task sufficiently well that there is seldom any need felt to duplicate its function.

Many a computing task can take the form of a *preprocessor* or *filter* through which one can pass a file to derive a useful variant. For example, string substitution (*define*) and library file inclusion (*include*) are useful operations on any source language; they provide a low level macro facility which is often all that is needed. A filter. Fortran is ugly but universal. We can still write in a "Rational Fortran" if we have a simple preprocessor that maps our pretty source into the standard language. Another filter. Even sorting can be performed as a filter, given sufficient scratch storage, in the guise of a stand alone process that reads all of its input once, then writes the sorted output in one swat.

A program that produces pretty listings, with titled and numbered pages, can be designed to work well with a broad class of languages. Similarly, a universal cross-referencer can be written (and has been, for the UNIX system[1] at Bell Labs). With tools like these, there is no longer a pressing need for large, multi-optioned compilers. Compiler writers can concentrate on the difficult enough task of mapping source code into object code (another filter!). And with the compiler-compilers available today to do the hard parts, there is no reason not to build a low-level mini-language for each new mini- or micro-computer that comes along. (Assembly language is sufficiently unreadable that it should be avoided whenever possible.) Indeed, a "Rational Fortran" preprocessor is just such a minicompiler.

The UNIX operating system makes filters all the easier to use by providing *pipes* to connect the output of one process, through a buffer, to the input of another simultaneously operating process. While this is not vital to the use of preprocessors, it does save the user from typing JCL and it saves the computer from writing numerous intermediate files which must then be deleted.

The important thing to observe is that each of the software tools mentioned here is a small coding project, ranging from a few man-months down to a matter of hours. Filters can be linked together in many useful ways, and some can be used for many different purposes, if they are designed from the start to interface to the world in standard ways. The programmer working this way becomes a tool builder, and his impact extends far beyond the task he may originally have set out to solve.

REFERENCE

1. Ritchie, D. M. and Ken L. Thompson, "The UNIX Time-Sharing System," *CACM* 17, 7, July 1974, pp. 365–375.

On being one's own programming self

by PETER J. DENNING
Purdue University
West Lafayette, Indiana

"Structured programming" has become the name of a popular movement toward better programming in the past two years.[1,2] Among the popular interpretations of the advice advanced by the masters, are many strong proclamations that anyone, by following a few simple prescribed rules, can be transformed into a programmer of top grade, and the productivity of programming shops everywhere can be multiplied. Experience, particularly with students in programming courses, leads to the tentative conclusion that some (but not all) will be helped by the "rules of structured programming;" and that the quality of their products will rise, though not to the extent one might expect from the inflated claims of some structuralists.

There is a striking similarity between writing good programs and writing good English.[3] When translated into the context of English Style, the counterparts of some structured programming rules border on the absurd. As examples, consider two specific directives one hears:

1. Construct all programs by combining simpler ones using only the control structures of "sequencing," "if-then-else," and "do-while."
2. Program top-down, successively refining "stubs" (unimplemented lower-level modules) until all modules are completed.

The problem with the first directive is that it has lost sight of the objective: to confine oneself deliberately to a small set of esthetically pleasing forms that permit clear expression without loss of generality. Just as many more than three prose forms are in good taste, many more than three programming forms are in good taste.[4,5,6] The great novelists and journalists have often achieved their statures precisely because they created a new form or a new combination of old forms. Even so in programming.

The second directive is more deceptive than the first. Once again, it loses sight of the objective: to structure a complex work so that its important elements can quickly be grasped in relation to the whole, and that it may easily and convincingly be presented. A hierarchical structure is a pleasing and effective way of achieving this objective. But the structure of the product need not resemble that of the process that created it. Even as the successful authors of large books have deviated from the advice of grammar school composition teachers—write first all the chapter titles, then all the section headings, then all the subsection headings, then all the topic sentences of paragraphs, then the paragraphs themselves—so the successful authors of large programs do not follow a particular, structured pattern of creativity: however quintessential in hierarchical design be their products. That is, in fact, the crucial point: a work of skill is judged by the beauty of its structure, though no one really believes that the craftsman's creative impulses conformed to any predetermined pattern. Each writer has his own style of creating his work and will achieve success only to the extent that he is able to find and cultivate his *own* style.

As you can see, the problem with both directives is their inflexibility. By saying, "Do it this way," they confine the individuality of the programmer and disallow the expression of his own creativity. Each writer—whether of prose or of program—has his own style of creating his works and will succeed only to the extent he is able to find and cultivate it.

Let it be clearly understood that I am not downgrading the movement toward better style in programming. I am one of its strongest supporters. By imitating the style of the masters I am able to instill more pleasing forms into my own writing and programming, and certain of the forms suggested by the masters are unquestionably effective. I am suggesting only that you beware dogmatic insistence on a fixed set of rules about programming. It is like insisting that students of composition write only in one style: if heeded, this advice would lead to a dreadfully dull world. Let us translate our experience with prose style into the context of programming. It may help us approach programming with the same realism, maturity, and flexibility as we approach prose.

REFERENCES

1. Dahl, O. J., E. W. Dijkstra and C. A. R. Hoare, *Structured Programming*, Academic Press, 1972.
2. Gries, D., "ACM Forum: On Structured Programming," *Comm. ACM 17,* 11 November 1974, pp. 655-657.
3. Kernighan, B. W. and P. J. Plauger, *The Elements of Programming Style,* McGraw-Hill, 1974.
4. Knuth, D. E., "Computer Programming as an Art," *Comm. ACM 17,* 12, December 1974, pp. 667-673.
5. Knuth, D. E., "Structured Programming with Goto Statements," *Computing Surveys 6,* 4, December 1974.
6. Strunk, W. and E. B. White, *The Elements of Style,* MacMillan, 1959.

Data types and program correctness*

by BARBARA H. LISKOV
Massachusetts Institute of Technology
Cambridge, Massachusetts

One of the most important current software issues is reliability, and accordingly, a major criterion of programming language design must be that the language contribute to the production of reliable programs. Although there are other important aspects of software reliability (e.g., fault tolerance), the most fundamental is program correctness: does the program do what it is supposed to do? A language can contribute to this goal by enhancing the provability of its programs. This paper discusses the impact of user-defined data types on program provability.

The principal motivation for having a language support user-defined data types is that they contribute to software reliability by enhancing the programmer's ability to use abstraction in writing programs. Abstraction plays an extremely important role in programming because it is the main tool available for controlling the complexity of programs. Thus the process of structured programming[1] is based primarily on recognition of useful abstractions. The abstractions provide a basis for problem decomposition: The original problem is solved by a program which uses the abstractions, and each abstraction becomes a new problem to be solved.

There are (at least) two kinds of abstractions used in programming: functional abstractions and data abstractions.[2] User-defined data types support the latter; in the absence of a type-definition facility, the use of data abstractions is at least difficult, if not impossible. However, many languages which permit new data types to be defined view new types as a name for some selected storage representation (e.g., Pascal,[3] Algol 68[4]). This is inadequate because it ignores the fundamental connection between a data type and a set of operations which are meaningful for that type. Instead the new type is manipulated by the operations which are meaningful for its representation. This means that the new type is not being used abstractly. Of course, procedures may be written to provide the meaningful operations, but these are defined separately from the type definition, so the relationship between them is not apparent.

The connection between data types and operations has been noted before, and recommended as a technique for treating types abstractly.[5,6,7] However, most work in this area has been concerned with identifying a distinguished set of operations which must be defined for every new type. For example, Balzer proposed a set consisting of four operations to access, update, insert, and destroy abstract data collections: each new type definition must specify how each of these operations is to be implemented.[6]

The problem with this approach is that there is no guarantee that the set of distinguished operations corresponds to the meaningful set of operations of the type. In fact, followers of this approach have noted that there are types which cannot be defined in terms of the distinguished set of operations.[6,7] As before, the additional operations could be defined by procedures, requiring the use of two separate mechanisms.

An approach which can handle any abstract data type by a single definition mechanism is to permit the programmer who identifies the data abstraction to define his own set of operations as part of the type definition. This requires that the language provide the proper type-definition mechanism: a type definition will consist of a description of how objects of the new type are to be represented in storage, and algorithms, written in terms of the representation, for all the operations.

One advantage of this approach is that all information about a type-definition is gathered in a single place and supported by a single construct (for example, the class construct of Simula 67[8]). This leads to simpler, more understandable programs. A more important advantage is that it is possible to encapsulate the type definition, so that users of the type can manipulate objects of that type only via the defined operations. In particular, the representation of objects is not visible outside the type definition. The type checking mechanism of the language can be used to enforce this. Languages containing such encapsulated mechanisms are under development.[9,10]

Encapsulating a type definition enhances the provability of programs because it permits proofs to be decomposed around the type definitions.* A single proof is given that the type definition implements the type correctly; this is the only proof in which implementation details must be considered. The proof of a program using the abstract type depends only on the abstract behavior of the type.

Consider the proof that a type definition correctly implements a type. This proof depends on the fact that en-

* This work was done with supp?ort of IBM funds for Research in Computer Science.

* Hoare makes this point and gives a sample of proof in Reference 11.

capsulation ensures that assumptions made by the operations about the meaning of the representation cannot be invalidated by actions outside the type definition. For example, the abstract data type, stack, with operations create, push, pop and empty, might be represented (in Pascal[3]) by

```
record  top:integer,
        data:array [1..100] of integer
end
```

Note that not every possible configuration of this structure is a legitimate stack; in legitimate stack representations, for example, $0 \leq top \leq 100$, and data [1], . . . , data[top] contain the pushed values, in the order in which they were pushed. The proof of the stack definition will consist of proofs that each operation behaves correctly, and each such proof assumes the operation is passed a legitimate stack object, and shows the operation returns a legitimate stack object. In the absence of encapsulation, proofs about the legitimacy of stack objects must be given in all programs using stacks, which is much more work.

The interface between the type definition and the users of the type must be precisely specified if this proof technique is to be successful. The encapsulation of the type definition ensures that the behavior of a type can be specified without describing how objects of the type are represented. For example, statements about the legitimacy of stack objects need not be included in a specification of the abstract type, stack. Thus encapsulation leads to simpler and more understandable specifications by reducing the information which must be expressed.[12]

The emphasis in the preceding paragraphs has been on program provability. Although this is interesting in its own right, it is also important because of its close relationship to program understandability. Understanding a program is the basis of an informal proof technique in which a programmer reasons about the meaning of a program (either his own or someone else's) in order to convince himself that it behaves correctly. The arguments advanced above concerning the relationship of data types to provability apply equally well to understandability.

REFERENCES

1. Dijkstra, E. W., "Notes on Structured Programming," in *Structured Programming*, ed. O.-J. Dahl, *et al.* A.P.I.C. Studies in Data Processing, No. 8. London, Academic Press, 1972, pp. 1–81.
2. Liskov, B. H. and S. Zilles, "Programming with Abstract Data Types," *ACM SIGPLAN Notices*, Vol. 9, No. 4, April 1974, pp. 50-60. (Also available as MIT Project MAC Computation Structures Group Memo 99)
3. Wirth, N. "The Programming Language PASCAL," *Acta Informatica*, V.1, 1971, pp. 35-63.
4. VanWijngaarden, A., B. J. Mailloux, J. E. L. Peck and C. H. A. Koster, "Report on the Algorithmic Language ALGOL 68," *Numerische Mathematik*, Vol. 14, 1969, pp. 79-218.
5. Mealy, G., "Another Look at Data," *Proc. AFIPS 1967 FJCC*, Vol. 31, pp. 525-534.
6. Balzer, R. M., "Dataless Programming," *Proc. AFIPS 1967 FJCC*, Vol. 31, pp. 557-566.
7. Earley, J., "Toward an Understanding of Data Structures," *CACM*, Vol. 14, No. 4, October 1971, pp. 617-627.
8. Dahl, O.-J., B. Myhrhaug and K. Nygaard. *The SIMULA 67 Common Base Language*, Publication S-22, Norwegian Computing Center, Oslo, 1970.
9. Liskov, B., *A Note on CLU*, Computation Structures Group Memo 112, MIT Project MAC, Cambridge, Ma., November 1974.
10. Wulf, W., *Alphard: Toward a Language to Support Structured Programs*, Department of Computer Science Internal Report, Carnegie-Mellon University, Pittsburgh, Pa., April 1974.
11. Hoare, C. A. R., "Proofs of Correctness of Data Representations," *Acta Informatica*, Vol. 1, 1972, pp. 271-281.
12. Liskov, B. and S. Zilles, "Specification Techniques for Data Abstractions," *Proceedings of the 1975 International Conference on Reliable Software*, Los Angeles, Ca., April 1975 (in press).

Extensibility in programming language design

by THOMAS A. STANDISH
University of California at Irvine
Irvine, California

INTRODUCTION

What is extensibility? What is it good for, if anything? Is it worth bothering about when designing a programming language?

Simply put, extensibility permits programming language users to define new language features. Starting with a *base language* and using various *definition facilities*,[1] an extensible language user can create new notations, new data structures, new operations, and, sometimes, new regimes of control. To a certain extent, extensibility permits a user to modify the features of a language to suit his changing needs and purposes. Given enough insight and craftsmanship, the user can create language extensions which are well-adapted to given intended application areas, and are useful for writing concise, clear algorithms free from contamination with low-level detail. However, experience has revealed that this approach is not as promising as was first hoped because of certain practical limitations I shall try to expose in this paper.

WHAT WERE THE EARLY ASPIRATIONS?

One of the earliest expressions of an extensible language philosophy was given by Brooker and Morris in 1960 in their classical paper on the Compiler-Compiler:[2]

> *The system is extendable and allows the user to define the meaning of new formats in terms of existing formats as well as in terms of basic assembly instructions (whose meaning is built in).*
>
> *It is unlikely that every machine user will want to write his own autocode: what is more likely is that he may wish to extend one of the standard languages to include statements suited to his own problem area.*

McIlroy seems to have had much the same idea in mind in 1960 when he explained his aim in introducing assembly language macros:[3]

> *It is our aim to show a limited set of functions readily implemented for a wide variety of programming systems which constitute a powerful tool for extending source languages conveniently and at will.*

In May of 1969, at the Extensible Languages Symposium, Christensen's chairman's introduction characterized the objectives of extensibility as follows:[1]

> *The ultimate and idealized objective of extensible languages is simple and attractive. A single universal programming system is postulated which is included in the software support for every general purpose computer. This programming system is not limited to one particular programming language such as PL/I. Rather, it includes a* base language *and a* meta-language. *A program in this system consists of, first, statements in the meta-language which expand, contract, or otherwise modify the definition of the base language to produce a* derived language, *and, second, statements in the derived language which constitute the executable part of the program.*
>
> *Thus the system includes facilities to define and then to program in a limitless variety of programming languages—languages which are used for business, scientific, or systems applications, languages which may be simple or complex.*

It seems fair to state that some of us were gripped by a curious euphoria in those days. A number of us believed that users would be able to extend the base of an extensible language rapidly and cheaply to encompass the data, operations, notation, and control natural to many diverse application areas. In short, we believed we could make it possible for unsophisticated users to manufacture personalized languages of reasonable efficiency and substantial utility with great ease just by applying some easily learned extension techniques. These beliefs were probably a bit over-ambitious.

EXTENSION TECHNIQUES—A BRIEF CATALOGUE

There are apparently quite a few dimensions to extensibility—more than we at first suspected. Let me give brief examples to illustrate the diversity.

Paraphrase

Paraphrase is a form of definition in which we define something new by showing how to exchange it for something whose meaning is known (or will become known after giving further definitions). This is commonly used in natural language to define the meaning of new words (e.g., thaumatology = the study or lore of miracles). Paraphrase is used widely as an underlying technique in extensible languages, and it takes many forms. As examples, consider the following:

(a) *macros* in assembly language. [3] E.g.,

```
MACDEF  SUM A, B, C
     LOAD  A
     ADD  B
     STORE  C
ENDDEF
```

(b) *procedure definitions* in algebraic languages. [4] E.g.,

```
integer procedure Factorial (N); integer N;
  Factorial := if N=0 then 1 else N*Factorial (N-1)
```

(c) *syntax macros*,[5] or grammatical extensions at the expression and statement level (as in Reference 2), E.g.,

```
smacro
  while ⟨b:boolean expression⟩ do ⟨s:compound
  statement⟩≡
    {create new label (Ll) in[
      Ll: if b then begin s; go to Ll end]}
```

(d) *data definitions*,[6,7] Here the idea is to be able to define new sorts of composite structured data starting with atomic data (such as *integers*, *characters*, and *reals*) or previously defined composite data. E.g.,

Let a *rational* be a structure which has
  a *numerator* which is an *integer* and has
  a *denominator* which is an *integer*
Let a *Matrix* be a *row* (*1*:*N*) with *vector* elements

(e) *operator definitions*,[8,9] E.g.,

Let Max(A,B) = (*if* A>B *then* A *else* B)
Let ++ be a *left associative*, *binary* operator with
  meaning = Max and *precedence* > "+"

(f) *control structure extensions*.[10,11] Here one can use process definition facilities to confer attributes on processes at the time of their definition. This is useful for defining processes that act as co-routines, that backtrack, that execute concurrently, which monitor for the occurrence of certain events and seize control when these events happen, and so forth.

Orthophrase

Orthophrase is a means of extension wherein we add "orthogonal" features to a language. An orthogonal feature is one which lies outside the space of features expressible by paraphrase. Its defining expression cannot be composed in the language. For example, suppose we are given a language *L* in which definition facilities (a)-(f) above are available but which lacks I/O device control primitives. We might not then be able to extend *L* by writing expressions in *L* itself to specify programs for reading disk files or printing on a line printer. Adding a file system, a real-time clock, a string valued function giving today's date, or adding call-by-reference parameter passing are four additional examples of the sort of features that cannot usually be defined by paraphrase if some basis for defining them is not already in the language. To add an orthogonal feature, one must normally perform surgery on the underlying guts of a language processor and patch it in. (The adjective "orthogonal" seems to have been borrowed by analogy to vector spaces, wherein a vector orthogonal to each of the vectors in a given basis set is one which can't be expressed as a linear combination of the basis set elements and which lies outside the space "spanned" by the basis elements).

Thus, orthophrase seems to be defined *relative* to a given set of paraphrastic definition facilities. It is not an absolute notion. Paraphrase facilities and an associated base language have a "span", so to speak, which consists of all possible paraphrastic extensions of the base. Any feature lying outside this "span" must be added by orthophrase, but a feature one must add by orthophrase in one language may be reachable by paraphrase in some stronger extensible language.

Metaphrase

While paraphrase and orthophrase add new capabilities, they do not change what is there before. *Metaphrase* consists of altering the interpretation rules of a language so that it processes old expressions in new ways. Examples of metaphrase are changing scoping policies for lifetimes or bindings of variables, changing the meaning of parameter evaluation, and changing the meaning of *assignment* and *go to* statements to allow programs to run backwards (useful perhaps in debugging to locate the source of an error after noticing incorrect program behavior).

WHAT HAVE WE LEARNED ABOUT EXTENSIBILITY?

By my latest count, 27 extensible languages have been proposed by 55 people (see Reference 12 for examples). Some of these proposed languages were implemented and, in some cases,[13,14] considerable experience has accumulated. What did we learn?

As you might expect, different categories of extensions can require widely different amounts of skill and labor. Some are hard and some are easy. Which kinds have which properties?

Let's first discuss extensions that can be made with modest amounts of labor by unsophisticated users. For example, in a well-designed extensible language,[15] it is straightforward to extend the real and integer arithmetic available in the base language to include rational, complex, or multi-precision arithmetic or to add matrix, vector, or formula manipulation. While these feats are easy, there is more in them than at first meets the eye.

For example, how do we arrange to share the syntax of arithmetic expressions among all these domains (so that

A+B*C could be used to specify operations on integers, matrices or complex numbers where appropriate)? How can we define arithmetic concisely on the huge space of mixed types (such as "a rational" + "a complex" or "a real" * "a matrix")? How do we augment the lexical analyzer to recognize and translate new expression forms for constants (such as "3+4i" for a complex constant)? Can we seal off the behaviors of underlying representations so users transact only with data types in the extension (such as preventing the user from tinkering with lower triangular arrays used as a substrate to represent symmetric matrices in the surface of the language)?

Other examples of extensions that can be done with a modest amount of labor include adding strings, lists, balanced binary trees and file records together with their appropriate operations, behaviors and notations. The extensible language experience [14] has revealed that not only must we be able to add typical combining, growth, and decay operators (such as concatenation, insertion, and deletion) and to express them using new or shared notations, we must also be able incrementally to extend common background language functions such as printing, assignment, and selection to handle special behaviors required of each new type (e.g., reduction of rational numbers to "least common terms" each time they are assigned to variables, or printing balanced binary trees in two dimensions if they will fit on a printed page, etc.).

In each of these extensions the "style" of the base language tends to be preserved unaltered. For example, none of these extensions affect conventions for the presence or absence of block structure, the presence or absence of declarations, or for the form of conditional and iterative expressions or for parameter evaluation.

Examples of hard extensions are: trying to add block structure to a language which doesn't have it (as in attempting to extend BASIC or FORTRAN to become ALGOL 60), adding declarations if they aren't there beforehand, adding backtracking or concurrent processing to the control structure if they aren't there already, or adding a real-time clock or a file system if they weren't there previously.

In oversimplified terms, the easy extensions seem to be those that use paraphrase to add new features, whereas the hard extensions seem to be those that use orthophrase or metaphrase to modify what was there before.

More precisely, in the case of paraphrase, the space of definitions users can give has been provided for explicitly in the extensible language design. Such extensions often specify independent additions to the base language and leave all base language features constant.

In the cases of orthophrase and metaphrase, one normally has to modify a description of the language processor. These descriptions are usually complex and considerable knowledge and sophistication are required of the user desirous of making alterations. It is often hard to add something new in an non-injurious way, so it blends smoothly and doesn't upset a collection of mutually dependent behaviors.

Here the extensible language experience reinforces the familiar notion that complex systems are resistant to change. The more intricate they are, the less easy it is to find out how to alter them significantly. Not only does this seem characteristic of the use of orthophrase and metaphrase, it also seems characteristic of cascades of extensions constructed by paraphrase—the more the intracacy of a set of extensions the more the difficulty of further extension.

These considerations seem to point to the conclusion that each extensible language is surrounded by an envelope of possible extensions reachable by modest amounts of labor by unsophisticated users. These easily reached extensions tend characteristically to be those which add one layer of new data and new operations and which perhaps make minor additions to notation while leaving the conventions of the base language invariant. Beyond this envelope, lie hard extensions requiring surgery on the language processor, major deformations of its conventions or style, or careful consideration of how to modify collections of mutually dependent extensions previously constructed. These seem generally to require a level of knowledge and sophistication beyond that of the casual user but perhaps attainable by the interested and committed professional. The unsophisticated user could no more likely add a file I/O facility to an extensible language not having one previously than he could add a date window to his wristwatch—both cases require a high level of investment in understanding complex descriptions and learning to use intricate tools. For all practical purposes, then, extensible languages seem only mildly extensible by unsophisticated users.

A basic difficulty with the philosophy of extensible languages derives from the fact that it is basically a "do-it-yourself-kit" philosophy. As a user, you may not want to build your own programming language any more than you may want to build your own car. Building your own car requires large investments of labor, knowledge, tools and time, and most home-builts are not very elegant. Unless you are a hobbiest, you are often willing to sacrifice control over form to obtain the benefits of prefabricated labor, especially when there is lots of choice in the marketplace. This is why most prefer to buy rather than build their cars.

So it is with extensible languages. It takes a huge dose of labor to extend a simple base to the level of capability represented by a high content language. Are users the right people to make these extensions or can skilled professional craftsman do a better job? Starting with a simple base and powerful extension facilities one often leaves the labor of building advanced features to those most likely to do an unattractive and unskilled job of it—the users.

A final difficulty revealed by the extensible language experience is that extending a simple base results often in long, thin extension cascades that are often ugly and inefficient. Look at how LISP[16] gets extended. One starts with a diamond (EVALQUOTE) and progressively corrupts it by adding warts (the PROG feature, SETQ, FUNARG, etc.). After you are finished, the extended LISP is far from simple and harmonious and you might have obtained better results by designing to meet the overall constraints all at once.

WHAT THEN IS EXTENSIBILITY GOOD FOR?

It should be obvious that the extensible language movement has succeeded only partially in meeting the objectives stated by the early explorers and that it did not create a programming revolution. It probably did succeed in clarifying for us the amounts of labor and knowledge required to produce several sorts of language variation, and in revealing what sorts of extensions are reasonable to expect of unsophisticated users.

Irons [17] probably put the matter in reasonable perspective when he observed that extensibility bears the same relation to high level languages as macros do to assembly languages. Maybe you use macros, say, 10 percent of the time or less, but they can be very convenient when you need them. Not only can you use them to suppress low level detail and promote program conciseness and clarity—you can often use them to mask irritating features of a language someone else has supplied. Extensibility seems worthwhile for analagous reasons. If implementation resources and execution space permit, it is probably better to have extensibility than to do without it.

REFERENCES

1. Christensen, C. and C. J. Shaw, *eds.*, Proceedings of the Extensible Languages Symposium, *SIGPLAN Notices*, August 1969.
2. Brooker, R. A. and D. Morris, "A General Translation Program for Phrase Structure Languages," *JACM*, January 1962, pp. 1-10.
3. McIlroy, M. D., "Macro Instruction Extensions of Compiler Languages," *CACM*, April 1960, pp. 214-220.
4. Naur, P. *et al.*, "Revised Report on the Algorithmic Language Algol 60," *CACM*, January 1963, pp. 1-17.
5. Leavenworth, B. M., "Syntax Macros and Extended Translation," *CACM*, November 1966, pp. 790-793.
6. van Wijngaarden, A. *et al.*, "Report on the Algorithmic Language Algol 68," *Num. Math.*, *14*, 1969, pp. 29-218.
7. Landin, P. J., "The Mechanical Evaluation of Expressions," *Computer Journal*, January 1964, pp. 308-320.
8. Taft, E. A. and T. A. Standish, *PPL User's Manual*, Tech. Rept., Center for Research in Computing Technology, Harvard University, September 1970.
9. Galler, B. A. and A. J. Perlis, "A Proposal for Definitions in Algol," *CACM*, April 1967, pp. 204-219.
10. Fisher, D. A., *Control Structures for Programming Languages*, Ph.D. Thesis, Carnegie-Mellon University, Department of Computer Science, May 1970.
11. Prenner, C. J., *Multi-Path Control Structures for Programming Languages*, Ph.D. Thesis, Center for Research in Computing Technology, Harvard University, June 1972.
12. Schuman, S., *ed.*, Proceedings of the International Symposium on Extensible Languages, *SIGPLAN Notices*, December 1971.
13. Irons, E. T., "Experience with an Extensible Language," *CACM*, January 1970, pp. 31-40.
14. Cheatham, T. E. and J. A. Townley, *Some Applications of ECL*, Center for Research in Computing Technology, Harvard University, 1973.
15. Wegbreit, B., *et al.*, *ECL Programmer's Manual*, Harvard Univ., 1973.
16. McCarthy, J., *et al.*, "Lisp 1.5 Programmer's Manual," *MIT Press*, Cambridge, 1962.
17. Irons, E. T., personnal communication, February 22, 1971.

Structured languages

by LEON PRESSER
University of California
Santa Barbara, California

A design process is a highly creative undertaking that has been generally guided by personal experiences and gross rules of design. This fact has been made explicit and at present the essence of the process of design is under investigation. This important problem is receiving attention in Computer Science. In particular, disciplined program design and development is the center of much discussion.

Conventionally a (software) design and development process commences with requirements analysis and proceeds through design, coding, testing, and finally the product moves into an operational environment and subsequent maintenance. The problem of error isolation and correction (i.e., debugging) is ubiquitous and the overall process is characterized by multiple iterations through various parts of the sequence of steps outlined above. In addition, appropriate documentation is generated throughout the project. The evaluation of this overall process is a complex issue that requires that 'evaluation' be a key objective from the very inception of the project through at least the beginning period of its operational lifetime. Currently, evaluation of design and development methodology is starting to receive attention but it may be some time before any substantial results are obtained. However, it is generally believed[1] that coding itself accounts for a relatively small portion of the total effort (⅙?).

Our objective here is to discuss structured (i.e., disciplined) languages and, with the context provided by the above paragraphs as a base, we can now proceed to highlight some of the important issues.

A structured language is one that, in addition to matching the needs of an intended application area, is designed to satisfy some specific objectives. Popularly, the main objective has been to produce a language that forces code to be written in a manner that prevents errors. This in turn implies clear documentation with a static version (i.e., listing) that resembles closely the dynamic behavior of the code. Another possible language design objective could be to have code written in such a manner that locality is explicit and performance in a virtual environment is improved. Still another goal might be to make testing more explicit. Let us concentrate on the error prevention objective.

Any program can be constructed with the following two control structures (Figure 1):

- *goto* (transfer of control forwards or backwards to any point)
- *predicate* (conditional transfer of control forwards or backwards to one of two possible points)

These two control statements represent a complete set for a programming language. However, the freedom provided by the potential use of the *goto* in an undisciplined fashion defeats the objectives previously outlined; in particular, it conflicts with the desire to have static and dynamic versions that are close. Consequently, the following complete set of control structures[2] is more appropriate for our purposes (Figure 2):

- *sequence* (transfer of control to next point in program sequence)
- *if-then-else** (conditional transfer of control to one of two operations which upon execution transfer control to next point following *if-then-else* in program sequence)
- *loop* (transfer of control so as to repeat an operation as long as some specified condition, which may be placed anywhere in the *loop,* is true; when the condition is no longer true control is transferred to next point following the *loop* in program sequence)

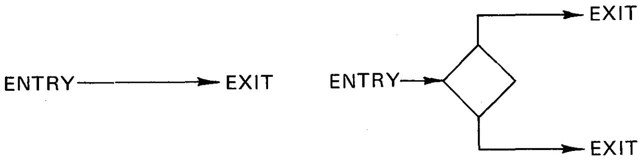
It is important to note that with this trio of control structures transfer of control occurs only in a disciplined manner. That is, with this set transfer of control always proceeds forwards in a strictly sequential fashion, possibly skipping the next operation in the sequence. Transfer of control backwards takes place only within the *loop* construct. Furthermore, each one of these structures possesses a single entry and a single exit point as shown in Figure 2. It is common to extend the *if-then-else* construct, which is a two-way branch, to an n-way branch called *case.*

We have thus far delineated the type of basic control structures that define a structured language. Additional discipline on the overall structure of programs, in order to facilitate readability and compile-time checks, may enforce the following format:

<program> ::= <declarations> <executable code>

Moreover, it is an objective of structured language design

* In principle, this structure is redundant since it can be obtained from *sequence* and *loop.* (The author is grateful to his student, Bob Haas, for bringing this point to his attention.)



GOTO

PREDICATE

Figure 1

to make basic programming language principles explicit. For example, scope and protection issues[3] may lead to the following program format:

```
<program> ::= <global declarations>
              <local declarations>
              <executable code>
```

In addition, a very desirable data definition facility may lead to the following format:

```
<program> ::= <definitions and declarations>
              <executable code>
<definitions and declarations> ::= <global section>
                                   <local section>
<global section> ::= <global new data structure
                      definitions>
                     <global declarations>
<local section> ::= <local new data structure
                     definitions>
                    <local declarations>
```

Furthermore, the important issue of attempting to enforce a design discipline through a structured programming language impacts on the final format of programs. This point is discussed in detail by White and Presser.[4] Hence, the overall structure of the programs produced with a structured language is shaped by the aggregate of objectives that determined the language design. In particular, it should be clear that a structured programming language may be used as a tool to enforce discipline that impacts well beyond the bounds of the coding phase. Let us now address the practical problem of obtaining a structured language.

There are three ways to generate a 'structured language', these are:

1. Imitation—Through the use of appropriately placed comments an existing language (e.g., Fortran) may be made to look like a language 'designed' around the three basic control structures previously described.

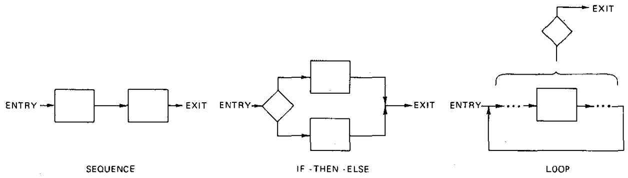


Figure 2

2. Preprocessing—A structured language may be designed such that programs written in it may be translated into an existing language for subsequent compilation and execution. In some cases this approach may hamper the features possible in the structured language.
3. Design—A structured language is obtained through design and direct implementation.

The last approach mentioned above is the most attractive one, however, as is commonly the case with any new programming language, vested interests are extremely difficult to overcome. On the other hand the importance of the issues under discussion is becoming clearer and the impact may be felt with some strength in the years ahead.

Finally, I would like to end by mentioning a couple of points of interest based on my personal experience in the direct design of a structured language. First, a programming language designed with objectives of structure tends to be almost free of ambiguities; this is due to the explicit treatment of principles and the inherent redundancies. Second, in the past I have used as a metric of the 'smoothness' of a programming language a count of the number of words like 'except', 'but', etc., present in the manual that describes how to use the language. The smoothness of a language appears to be indirectly proportional to this number, and a structured language tends to be quite smooth!

REFERENCES

1. Brooks, F. P., Jr., "The Mythical Man-Month," *Datamation, 20,* 12, December 1974.
2. Böhm, C., and G. Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules," *Comm. ACM 9,* 5, May 1966.
3. Presser, L., and J. White, "Making Global Variables Beneficial," *Proceedings IFIPS,* August 1974.
4. White, J., and L. Presser, "A Tool for Enforcing System Structure," *Proceedings ACM National Conference,* August 1973.

Structured control in programming languages*

by CHARLES T. ZAHN, JR.
Stanford Linear Accelerator Center
Stanford, California

CONCEPTUAL DISTANCE

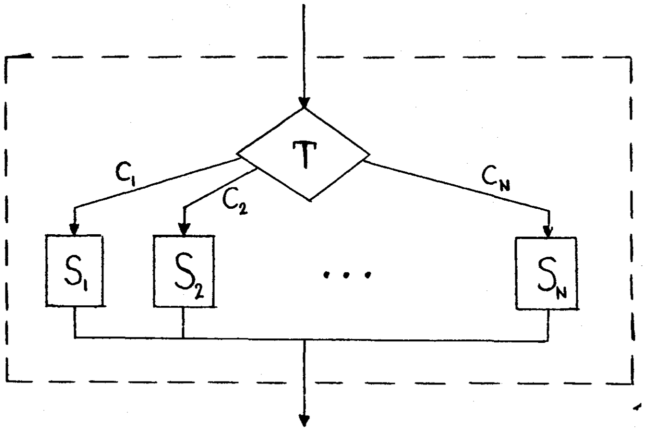
Solving a problem with the aid of a computer involves the construction and execution of a program described by a linear piece of text. First, the problem-solver (programmer) translates his problem into a procedural solution embodied in a static program text, written in a programming language. Then a computer is caused to perform a dynamic sequence of actions in accordance with the commands in the program text. The reliability of this two-stage problem solution (i.e., the likelihood that the actions performed really provide a solution of the problem) depends on the degree to which the program text mirrors the possible action sequences that it causes,[1] as well as the problem solution that it purports to implement. It is useful to speak of the "conceptual distance" between program text and action sequences or between problem definition and program text. The programmer who wants some measure of confidence in the reliability of his program must bridge both these conceptual distances. It follows that a major goal of programming language design should be to help reduce both these distances.

STRUCTURED CONTROL

Structured programming[1] is a systematic step-wise method of program composition which can be used to conquer the distance between problem and program by chopping it into bite-sized pieces and employing abstraction as a mental aid to control the problem of complexity. It reduces the distance between program text and action sequence by employing in the program text only those forms of sequence control which allow an easy visualization of the possible action sequences from an inspection of the static linear program text. The control structure includes sequential grouping of commands as well as command selection (*if* statements and *case* statements) and repetition (*while* and *repeat* statements). Enumerative reasoning and mathematical induction are available mental aids for understanding the action sequences evoked by programs restricted to these forms of sequence control. These considerations suggest a control structure limited to sequential grouping, selection and repetition.

PROBLEM-ORIENTED CONTROL

Unfortunately, the story doesn't end there because, in spite of the immense advantages of the restricted control, it is still not adequately problem-oriented. This is true even when the control structure is extended by a simple *for* statement and recursive procedures and functions. One of the commonest situations in programming is the need to select one of a finite set of commands, using some selection mechanism, each of whose outcomes corresponds to a unique command from the set. The following general flowchart models the control:



In this flowchart, $C_1,C_2,\ldots,C_N$ are constants of some finite type, each S_k is a command (statement), and T is a "test" or inspection of program variables whose execution terminates by selection of one of the C_k as its outcome.

The special case ($N=2$, $C_1=true$, $C_2=false$ and T evaluates a logical expression B) represents the familiar control form *if* B *then* S_1 *else* S_2. The case ($C_k=k$ for $1\leq k\leq N$ and T selects that constant C_k equal to the value of an integer expression E) represents Hoare's integer case statement with the syntax *case* E *of* $(S_1;S_2;\ldots;S_N)$. This expression-driven case statement has been generalized[2] and implemented in PASCAL.[3] By allowing constants C_k and expression E to correspond to any finite type (espe-

* This work supported by the U.S. Atomic Energy Commission under contract AT(043)515.

cially programmer defined types like Color whose 4 constant values might be Green, Blue, Red, Black), the conceptual distance between problem and program can be greatly reduced.

There remain situations in which the selection is not conveniently reduced to an expression evaluation, and T must be a compound command which returns a value C_k. It also naturally occurs that at certain places within T, it becomes clear which value should be selected and an immediate termination of T is entirely appropriate. Recent versions[4] of the programming language BLISS[5] extend the restricted control by allowing any compound statement to be labeled; then a statement of the form *leave L with E*, causes immediate termination of the enclosing statement labeled L and returns E as its value. It is, therefore, easy to implement the more general selection mechanism T within BLISS. A more recent proposal[6,7] for extending the control is an event-driven case statement of the form

$$\textit{until } C_1 \textit{ or } C_2 \ldots \textit{ or } C_N \textit{ do } T \textit{ then case}$$
$$(C_1{:}S_1; \ldots ;C_N{:}S_N)$$

with event statements C_k within T, causing immediate termination of T and selection of C_k. Each C_k is an identifier or name created by the programmer to provide a problem-oriented description of what the program is doing. The syntax for this generalized case statement was motivated by considerations of writing and reading programs in a top-down fashion. The number of similar proposals for a termination mechanism (see the survey by Knuth[7]) shows the universal need for such a programming device. Other common situations requiring an explicit termination mechanism are repetitions of a command sequence where the detection of the termination condition naturally occurs midway through the sequence and the handling of error conditions which have various degrees of severity.

REPETITIONS WITH A CONTROL VALUE

It is a common need in programming to repeat a given compound command once for each of a well-defined finite sequence of values, where that value is accessible (but not changeable) within the repeated command. When the programmer's intent is exactly reflected in this special form of repetition, there is a great gain in clarity when the program text employs a special syntax to indicate the repetitive pattern. Certainly, there should be a repetition like

$$\textit{repeat for } V \textit{ from } E_1\{\textit{upthru/downthru}\}E_2 \textit{ do } S(V)$$

where V is a variable of ordered finite type and E_1,E_2 are expressions of that type. This is the form (with slight differences in syntax) of *for* statement implemented in PASCAL.[3,8]

Serious consideration should be given to extensions of the *for* statement to cater for progressions of values defined by more general successor functions. For example, the programmer who builds sequences using *records* and *references* is helped immensely by statements like

$$\textit{repeat for } R \textit{ from } \text{Start } \textit{by } \text{Next } \textit{upto null do } S(R)$$

where R is a reference variable whose values are Start, Next (Start), Next (Next(Start)), etc., up to but not including *null*. The use of words *upthru, downthru, upto* is an attempt to reduce the ambiguity that results from not making explicit the distinction between inclusion or exclusion of the final item.

PROCEDURAL MECHANISMS

Procedures and functions, with carefully designed parameter mechanisms, are now more widely appreciated as beneficial tools for program decomposition and the embodiment of problem-oriented abstractions. They are thus helpful to the programmer in his task of bridging the conceptual distance between problem and program; that is, when their use is not discouraged by considerations of efficiency. The programmer should be allowed to attach the *macro* option to any procedure or function invocation, and thereby feel free to use them as purely structuring tools without the run-time overhead often implied by the closed subroutine.

The main difficulty in the use of procedures and functions is that the conceptual distance between program text and dynamic actions is often increased by mysterious parameter mechanisms and side-effects.[9] The axiomatic definition of *procedures* and *functions* in PASCAL[8] can be interpreted as a suggestion that procedure parameters be classified as *constant* or as *update*, while function parameters are restricted to *constant*. A *constant* parameter represents a constant value determined by an actual parameter expression at the time the procedure or function is invoked. This value may not be altered by the procedure or function. This has usually been referred to as "call by value". An *update* parameter represents a program variable whose value can be altered or inspected by the procedure. The actual variable being inspected and altered is the one whose name is given as the actual parameter in the procedure invocation. It would probably be an aid to program clarity to distinguish a third class of *result* parameters which may not be inspected (since they are presumably as yet undefined!), but which are expected to be assigned values by the procedure. Of course, *result* parameters would not be allowed for functions.

The program text of a procedure or function should indicate all those global (i.e., non-local, non-parameter) variables which are referenced within it with a textually clear distinction of those which are potentially alterable by the procedure. No functions should alter any globals. Whether this additional program documentation is made the responsibility of the programmer or a helpful compiler—in either case it provides crucial textual evidence to aid the programmer in visualizing the possible dynamic actions caused by a given invocation of the procedure or function. Another important restriction[8] is the disjointness of the set of alterable parameters and global variables. Failure to comply with this restriction may cause very nasty and subtle errors.

It has recently been proposed by Hardgrave[10] that a keyword,* rather than positional notation for the correspondence between formal and actual parameters, would have several nice advantages, one of which is the obvious textual clarity of the programmer's intent. In the case of procedures and functions with long parameter lists, there is a disturbing potential for erroneous parameter communication even in a highly typed language. By allowing *default* actual parameters[10] for certain formal parameters to be explicitly given within the procedure declaration, the textual length of the invocation can often be kept reasonably small in spite of the apparent verbosity of the keyword notation. It is also possible to add a new parameter without altering previously written invocations of the procedure—a potentially non-trivial advantage in a large software project requiring modifications through time.

Recursive *procedures* and *functions* should be allowed since they reflect problem solutions whose reprogramming without recursion involves considerable conceptual distortion and, therefore, increases the conceptual distance between problem and program. In a similar way, there are certain problems which are most naturally solved by two or more procedures whose relationships to one another are more symmetric than the normal hierarchical procedure relationships.[11] Such procedures are known as coroutines or semi-coroutines and they differ from normal procedures in that each time they are invoked from another coroutine they resume execution where they last left off. Their cooperative behavior is understandable in terms of an anthropomorphic model in which each coroutine is executed by a different person who simply goes to sleep when he *resumes* one of the other coroutines, but when his own coroutine is resumed again he awakens in the same state as before he went to sleep. Coroutines can be used to obtain the conceptual advantages of a multi-pass algorithm without the actual need for secondary storage and data format specifications usually implied by a literal implementation of the separate passes.[12] Especially compelling examples of the conceptual correctness of coroutines are to be found in Dahl.[11] The coroutines discussed here are never in simultaneous or interleaved execution so their correct behavior doesn't involve the deeper problems of mutual exclusion, deadlock, etc.

* The correspondence between an actual and formal parameter is indicated textually by <formal parameter> = <actual parameter>.

CONCLUSION

An attempt has been made to discuss various issues involved in the design of control for a programming language by relating these design issues to the goal of reducing "conceptual distance". A slight compromise to the strict structured control seems justifiable to obtain a more problem-oriented control. More research would be worthwhile in the area of "safe" iterations, parameter mechanisms and coroutines.

REFERENCES

1. Dijkstra, E. W., "Notes on Structured Programming," in *Structured Programming* by O. J. Dahl, E. W. Dijkstra and C. A. R. Hoare, 1972, Academic Press, London and New York.
2. Hoare, C. A. R., "Notes on Data Structuring," in *Structured Programming* (see 1).
3. Wirth, N., "The Programming Language PASCAL," *Acta Informatica*, Vol. 1, No. 1, pp. 35-63.
4. Wulf, W. A., "A Case Against the *goto*," *Proc. ACM National Conference*, 1972, pp. 791-797.
5. Wulf, W. A., D. B. Russell and A. N. Haberman, "BLISS: A Language for Systems Programming," *CACM*, Dec. 1971, pp. 780-790.
6. Zahn, C. T., "A Control Statement for Natural Top-down Structured Programming," presented at Symposium on Programming Languages, Paris, 1974, Springer-Verlag.
7. Knuth, D. E., "Structured Programming with *goto* Statements," *ACM Computing Surveys*, December 1974.
8. Hoare, C. A. R. and N. Wirth, "An Axiomatic Definition of the Programming Language PASCAL," *Acta Informatica*, 1973, pp. 335-355.
9. Hoare, C. A. R., *Hints on Programming Language Design*, Stanford University Computer Science Department Report No. CS-403, October 1973.
10. Hardgrave, W. T., *Positional versus Keyword Parameter Communication in Programming Languages*, Report of the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, Virginia, September 1974.
11. Dahl, O. J., "Hierarchical Program Structures," in *Structured Programming* (see 1).
12. Knuth, D. E., *The Art of Computer Programming*, Volume 1, Chapter 2, 1968.

Issues in programming language design— An overview*

by ANTHONY I. WASSERMAN
*University of California at San Francisco***
San Francisco, California

The past few years have witnessed an increased understanding of the relationship between programming languages and problem solving. Programming is now understood to be a rather difficult task which requires the simultaneous application of principles, skills, and art.[1,2,3] Computer scientists have recognized that the features of a programming language can have a significant effect upon the ease with which reliable programs can be developed. It has also been observed that certain languages and language features are particularly well suited for the use of systematic programming techniques, while others hinder or discourage such discipline.[4,5,6] Of course, it is possible to write well-structured, clearly organized programs in any programming language, but such programs have often been the exception rather than the rule.

As a result of this work, there have been numerous developments in the general area of programming languages. Among these are the following:

(1) a significant number of new programming languages have been designed and/or implemented,[7] with several developed principally to promote proper programming practices;[8,9,10]
(2) the general features of existing and proposed languages have been analyzed in an attempt to identify desirable characteristics of programming languages;[11,12,13,14] strong criticism has been directed at those languages which do not appear to contain the requisite features for the systematic development of reliable software;[15,16]
(3) preprocessors have been implemented for several programming languages, thereby allowing programmers to use "structured programming" techniques;
(4) direct modifications have been designed and/or implemented for several programming languages, in order to enhance their suitability for program development;
(5) general design criteria for programming languages have been advanced, with attention focused on the need for a language to have a sound theoretical basis;[17,18]

Although the design goals for the individual language modifications and language developments vary considerably, there are a number of common objectives which can be identified. First, the value of linear flow of control was recognized, primarily for its value in program debugging and verification, and powerful control structures were proposed and added to promote such a flow.[19,20,21,22] Second, the value of abstraction was recognized as a way to develop a representation of information which is more closely related to the application being programmed than exists in any programming language with a fixed number of data types.[23,24] Third, the scope and binding of variables was studied as a technique which simplifies program verification and which reduces programming errors caused by side effects.[25,26] Fourth, it was recognized that a language must be comprehensible, so that programs written in the language can be read and maintained. Fifth, efforts were made to limit the size of languages, in order to make them easier to implement and to make it possible for a programmer to thoroughly understand the tool. Finally, modular program structures were observed to make an important contribution to the production of large software systems.

These design objectives are reflected in a variety of decisions which are made in designing programming languages. Since the universe of design objectives is somewhat self-contradictory, as is immediately evident from a comparative analysis of languages, the language designer must consider the tradeoffs among the various possible features for a language, and give more emphasis to some of these objectives than others. It is agreed, however, that the language designer must have a thorough understanding of the goals of the language prior to commencing a specification of the syntax and semantics of the language.

Although there are a large number of closely related issues involved in the design of a language, much of the current work in language design is focused on three areas: language extensibility, data types and abstraction, and control structures.

Language extensibility refers to the ability of the programmer to modify the language being used, with the intent of extending the power of the language.[23,27,28] A relatively small "base" language is defined, along with capabilities to add features such as new data types, new

* This work was supported in part by the Commonwealth Fund.

** Graduate Program in Medical Information Science. The author is also affiliated with the Computer Science Division, University of California at Berkeley.

operators, new syntax, and new control structures in order to enable the program to more closely correspond to the problem domain.[29] The derived language or "task language" which thereby results can allow programs to be written in such a way that they are comprehensible to almost anyone familiar with the application area of the program. Persons working on the development of extensible languages foresee the establishment of a higher level language which could evolve gracefully via packages of definitions. The availability of such packages for particular task areas could then greatly increase programmer productivity.

Data types and abstraction refers to the number of predefined data types which are available in a language, the means available for combining the primitive types to create more complex types, and the way in which new abstractions may be introduced into the language.[24] The notion of an abstract data type has been advanced to define a class of abstract objects which is completely characterized by the operations available on those objects.[30] The means by which a variable takes on a given type and has a value assigned to it are extremely important issues in language design.[31,32]

Control structures are the means by which the order of execution of statements in a program is determined. While it has been formally shown that only sequential control, a conditional statement, and iterative control are necessary to describe any computation,[20] it is also understood that restricted use of the *go to* statement may result in greater program clarity.[33] Much of the work in control structures has dealt with the definition of mechanisms for conditional testing and iteration which reduce the need for the *go to* statement, and which produce dynamic program behavior closely resembling the static program structure. Because of the need to permit communication among tasks, various control structures have been proposed which permit coroutines,[34] parallel processing, synchronization, and monitoring.[35,36,37] A wide variety of proposals for improving control structures of existing and new languages have been suggested, including forms of nondeterministic control,[38,39] and the relative power and merits of these alternatives have been discussed extensively.

Although these three issues are at the heart of much of the work on programming language design, there are a number of other issues which have received attention. First, the rapid growth of interactive systems and their use by non-programmers has identified a need for string processing facilities and exception handling capabilities.[40,41] Second, the development of conversational programs for access to large data bases has focused attention on the need for capabilities in the area of data management and the need for more powerful input/output facilities.[42] Third, there are standardization efforts in progress for a number of programming languages in order to improve program transferability. Fourth, research into program proving and verification has led to additional proposals for programming languages.[43] Finally, there is also a need to simplify the task of program documentation, so that one can easily understand how a program works.

The language designer must then be able to synthesize all of these various concepts in such a way as to produce a language which is defined in a uniform way, which has a logical relationship between the syntax and the semantics, which allows an efficiently executing program to be produced, and which permits programmers to conceptualize a solution to a problem in a straightforward manner. The interrelationships among these design criteria are extremely complex, and it appears that it will be some time before a language emerges which can satisfy all of the needs of a broad class of programming applications.

Beyond that point, there are a number of political and economic issues which will affect the eventual acceptance of such a language. The primary determinants appear to be the support given to the language through implementation by major vendors of computer hardware and software, and the ease by which programmers and programming management can be attracted away from their present language and trained in the new language. Until then, most programmers will be left to work with tools which are now recognized to be somewhat inadequate for the effective solution of programming problems.

In conclusion, then, several key questions can be raised concerning the design of programming languages. How do we develop a programming mechanism which can accurately mirror logical thinking?[44] Furthermore, how do we develop a tool which is suitable for stepwise refinement of the problem from its abstract form to its "elaborated" form in a "natural" way? Last, how then does such a language get introduced and accepted by the general programming community so that it raises the quality of software production? These are the main questions which underlie present research and development in the field of programming languages.

REFERENCES

1. Dijkstra, E. W., "The Humble Programmer," *CACM,* Vol. 15, No. 10, October 1972, pp. 859-866.
2. Knuth, D. E., "Computer Programming as an Art," *CACM,* Vol. 17, No. 12, December 1974, pp. 667-673.
3. Dennis, J. B., "The Design and Construction of Software Systems," in *Advanced Course on Software Engineering,* ed. M. Beckmann *et al.* Berlin: Springer Verlag, 1973, pp. 12-28.
4. Dahl, O-J., E. W. Dijkstra and C. A. R. Hoare, *Structured Programming,* London: Academic Press, 1972.
5. Dijkstra, E. W., "GOTO Statement Considered Harmful," *CACM,* Vol. 11, No. 3, March 1968, pp. 147-8.
6. Wulf, W. A., "A Case Against the GOTO," *ACM SIGPLAN Notices,* Vol. 7, No. 11, November 1972, pp. 63-69.
7. Sammet, J., "Roster of Programming Languages for 1973," *Computing Reviews,* Vol. 15, No. 4, April 1974, pp. 147-160.
8. Wirth, N., "The Programming Language PASCAL (Revised Report)," Berichte der Fachgruppe Computer-Wissenschaften, Eidgenossische Technische Hochschule, Zurich, 1973.
9. Wulf, W. A., "BLISS: A Language for Systems Programming," *CACM,* Vol. 14, No. 12, December 1971, pp. 780-790.
10. Liskov, B., "A Note on CLU," Computation Structures Group Memo 112, MIT Project MAC, 1974.
11. Cheatham, T. E., Jr., "The Recent Evolution of Programming Languages," *Proceedings IFIP Congress 71,* Amsterdam: North-Holland, 1972, pp. 298-313.

12. Elson, M., *Concepts of Programming Languages,* Palo Alto: Science Research Associates, 1973.
13. Ledgard, H. F., "Ten Mini-Languages: a Study of Topical Issues in Programming Languages," *Computing Surveys,* Vol. 3, No. 3, September 1971, pp. 115-146.
14. Wasserman, A. I., "Online Programming Systems and Languages: a History and Appraisal," University of California at San Francisco Laboratory of Medical Information Science Technical Report No. 6, July 1974.
15. Aiello, J. M., "An Investigation of Current Language Support for the Data Requirements of Structured Programming," MAC Technical Memorandum 51, MIT Project MAC, 1974.
16. Sherertz, D. D., A. I. Wasserman and D. R. Allison, "Some Critical Comments Concerning MUMPS," *Proceedings 1974 MUMPS Users' Group Meeting,* St. Louis: MUMPS Users' Group, Biomedical Computing Laboratory, Washington University, pp. 173-190.
17. Wirth, N., "On the Design of Programming Languages," *Information Processing 74.* Amsterdam: North Holland, 1974, pp. 386-393 (preprints).
18. Hoare, C. A. R., "Hints on Programming Language Design, " Stanford University Computer Science Department Technical Report CS-73-403, December 1973.
19. Fisher, D. A., "A Survey of Control Structures in Programming Languages," *ACM SIGPLAN Notices,* Vol. 7, No. 11, November 1972, pp. 1-13.
20. Bohm, C. and G. Jacopini, "Flow Diagrams, Turing Machines, and Languages with only Two Formation Rules," *CACM,* Vol. 9, No. 5, May 1966, pp. 366-371.
21. Herriot, R., "A Uniform View of Control Structures in Programming Languages," *Information Processing 74,* Amsterdam: North Holland, 1974, pp. 331-335 (preprints).
22. Zahn, C. T., "A Control Statement for Natural Top-Down Structured Programming," in *Programming Symposium: Proceedings, Colloque sur la Programmation,* ed. B. Robinet. Berlin: Springer-Verlag, 1974, pp. 170-180.
23. Cheatham, T. E., Jr., "Motivation for Extensible Languages," *ACM SIGPLAN Notices,* Vol. 4, No. 8, August 1969, pp. 45-48.
24. Flon, L., "A Survey of Some Issues Concerning Abstract Data Types," Carnegie-Mellon University, Department of Computer Science Technical Report, September 1974.
25. George, J. E. and G. R. Sager, "Variables—Bindings and Protection," *ACM SIGPLAN Notices,* Vol. 8, No. 12, December 1973, pp. 18-29.
26. Presser, L. and J. R. White, "Making Global Variables Beneficial," *Information Processing 74,* Amsterdam: North-Holland, 1974, pp. 413-418 (preprints).
27. Galler, B., "Extensible Languages," *Information Processing 74,* Amsterdam: North Holland, 1974, pp. 313-316 (preprints).
28. Schuman, S., (ed.) Proceedings of the International Symposium on Extensible Languages, *ACM SIGPLAN Notices,* Vol. 6, No. 12, December 1971.
29. Wegbreit, B., "The ECL Programming System," *Proceedings AFIPS 1971 FJCC,* Vol. 39, pp. 253-262.
30. Liskov, B. and S. Zilles, "Programming with Abstract Data Types," *ACM SIGPLAN Notices,* Vol. 9, No. 4, April 1974, pp. 50-60.
31. Morris, J. H., Jr., "Types are not Sets," *Conference Record of ACM Symposium on Principles of Programming Languages,* 1973, pp. 120-124.
32. Wegbreit, B., "The Treatment of Data Types in EL1," *CACM,* Vol. 17, No. 5, May 1974, pp. 251-264.
33. Knuth, D. E., "Structured Programming with *GOTO* Statements," *Computing Surveys,* Vol. 6, No. 4, December 1974, pp. 261-301.
34. Conway, M. E., "Design of a Separable Transition-Diagram Compiler," *CACM,* Vol. 6, No. 7, July 1963, pp. 396-408.
35. Brinch Hansen, P., *Operating System Principles,* Englewood Cliffs, Prentice-Hall, Inc., 1973.
36. Dijkstra, E. W., "Hierarchical Ordering of Sequential Processes," in *Operating System Techniques,* ed. Hoare and Perrott, London: Academic Press, 1973, pp. 72-93.
37. Hoare, C. A. R., "Monitors: an Operating System Structuring Concept," *CACM* Vol. 17, No. 10, October 1974, pp. 549-557.
38. Floyd, R. W., "Nondeterministic Algorithms," *JACM,* Vol. 14, No. 4, October 1967, pp. 636-644.
39. Dijkstra, E. W., "Guarded Commands, non-determinacy, and a Calculus for the Derivation of Programs," Report EWD418, Burroughs, Nuenen, the Netherlands, 1974.
40. Wasserman, A. I., "Some Principles of User Software Engineering for Information Systems," *IEEE COMPCON Spring 75 Conference Digest* (in press).
41. Goodenough, J. B., "Structured Exception Handling," *Conference Record of the Second ACM Symposium on Principles of Programming Languages,* 1975, pp. 204-224.
42. Codd, E. F., "Recent Investigations in Relational Data Base Systems," *Information Processing 74,* Amsterdam: North-Holland Publishing Co, 1974, pp. 1017-1021 (preprints).
43. Kosy, D. K., "Approaches to Improved Program Validation through Programming Language Design," in *Program Test Methods,* ed. W. Hetzel. Englewood Cliffs: Prentice-Hall, Inc., 1973, pp. 75-92.
44. Wirth, N., "On the Composition of Well-Structured Programs," *Computing Surveys,* Vol. 6, No. 4, December 1974, pp. 247-260.

An overview of the 1974 COBOL standard

by MARGARET M. COOK, GEORGE N. BAIRD, WILLIAM M. HOLMES, PATRICK M. HOYT, L. ARNOLD JOHNSON and PAUL OLIVER

Department of the Navy
Washington, D.C.

INTRODUCTION

Since 1 July 1972, all COBOL compilers brought into the Federal Government have to be identified as implementing one of the levels of the Federal COBOL Standard. The National Bureau of Standards, which has the responsibility for the development and maintenance of Federal ADP Standards, has delegated to the Department of Defense the responsibility for the operation of a Government-wide COBOL Compiler Testing Service. This responsibility is discharged by the Federal COBOL Compiler Testing Service (FCCTS), an activity of the Department of the Navy's Automatic Data Processing Equipment Selection Office Software Development Division, through the implementation and maintenance of the COBOL Compiler Validation System (CCVS),[1] a comprehensive set of computer programs used to test COBOL compilers for compliance with the Federal COBOL Standard.

In May 1974, the American National Standards Institute approved ANS Programming Language COBOL, X3.23-1974[2] as the national standard for the COBOL language replacing USA Standard COBOL, X3.23-1968.[3] Federal Information Processing Standards Publication 21-1[4] adopts X3.23-1974 (minus the Report Writer module) as the Federal COBOL Standard. As a result of these actions, the Testing Service is engaged in the development of a new COBOL Compiler Validation System, incorporating tests for the revised language. This paper presents an overview of COBOL 74, highlighting the new features in the language, major language deficiencies, and important contributions to the programming discipline. All comments made with regard to COBOL 68 or COBOL 74 will be based on the language as defined in References 3 and 2, respectively.

REVISIONS TO THE COBOL 68 MODULES

The Nucleus, Sequential I-O and Library modules of COBOL 68 have undergone major revisions. The Table Handling and Segmentation modules are essentially the same modules as appeared in the 1968 COBOL Standard. The Random Access module has been replaced by the new modules, Relative I-O and Indexed I-O, and the specifications for the Report Writer module have been rewritten. The Sort module has been expanded into the Sort-Merge module by the addition of the MERGE verb.

The Table Handling module in COBOL 74 consists of two levels instead of the previous three levels, and the punctuation changes in the 74 Standard have relaxed the rigid rules for the use of parentheses and commas in referencing table items. Literals and index-names may be mixed when referencing a table item, and an index may be incremented or decremented by a negative value.

In the remainder of this section we concentrate on the new features of the Nucleus and Library modules. The I-O modules will be covered in a later section, and we ignore the Report Writer as being beyond the scope of the Federal Standard.

Section 2.3 of Appendix B of the X3.23-1974 Standard[2] documents in greater detail the modifications made to the 1968 Standard and the additional language features added for the 1974 Standard.

Nucleus

Three new verbs have been added to the Procedure Division of the Nucleus module; INSPECT, STRING and UNSTRING. The INSPECT verb replaces the old EXAMINE verb and provides expanded editing capabilities. With an INSPECT statement one can count, and replace occurrences of either single characters or groups of characters in a data item. For each of these INSPECT functions, the BEFORE or AFTER phrase allows one to specify that the function begins or ends upon encountering a given character or group of characters.

As an example of the INSPECT statement, consider the following source code. (Coding examples in this paper will contain formatting errors due to the typesetting requirements.)

DATA DIVISION entries

```
01    ID-1 PIC X(54) VALUE 'b̸b̸DEFEATb̸b̸b̸b̸b̸b̸
      DEDUCTb̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸
      DEFENSEb̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸DETAIL'.
```

PROCEDURE DIVISION entries

```
INSPECT ID-1 REPLACING ALL 'b̸b̸DE' by 'THEb̸',
  FIRST 'A' BY 'E',
  FIRST 'b̸b̸b̸b̸' BY 'b̸OFb̸',
  FIRST 'T' BY 'K' AFTER INITIAL 'UC',
  FIRST 'b̸b̸b̸b̸b̸b̸b̸b̸b̸'  BY 'b̸GOb̸OVERb̸',
    FIRST 'S' BY 'C',
    FIRST 'b̸b̸b̸b̸b̸b̸b̸b̸' BY 'b̸BEFOREb̸' AFTER INITIAL
    'EN'.
```

After executing the above INSPECT statement ID-1 would contain

```
'THEb̸FEETb̸OFb̸THEb̸DUCKb̸GOb̸
             OVERb̸THEb̸FENCEb̸BEFOREb̸THEb̸TAIL'.
```

This example points out one of the limitations on the use of INSPECT; the number of characters being replaced must equal the number of characters by which they are replaced. Thus, 'DE' cannot be replaced by 'THEb̸' but the characters 'b̸b̸DE' can be replaced by 'THEb̸'.

With the 1974 Standard, the COBOL language has the features necessary for manipulation of character strings without each individual character string beginning in the leftmost character position of a data item. The STRING statement allows a user to build a single data item from two or more data items. The sending data items may be delimited by one or more character(s), or the entire sending item can be part of the receiving item. A user can also indicate the relative starting position in the receiving item for the STRING operation.

The following example illustrates the use of the STRING statement.

DATA DIVISION Entries

```
01   WISH-LIST.
     02   FILLER PIC X(4) VALUE 'GEEb̸'.
     02   FILLER PIC X(33) VALUE SPACES.
01   REL-POSITION PIC 99 VALUE 5.
01   STRING-VALUES.
     02   FIELD1 PIC X(13) VALUE 'Ib̸WISHb̸I,b̸YOU'.
     02   FIELD2        PIC        X(24)        VALUE
          'WASb̸Ab̸FORTRANb̸PROGRAMMER'.
```

PROCEDURE DIVISION Entries

```
PARAGRAPH-1.
      STRING FIELD1 DELIMITED BY ', ',
          SPACES, FIELD2 DELIMITED BY SIZE
          INTO WISH-LIST WITH POINTER REL-PO-
          SITION,
          ON OVERFLOW GO TO PARAGRAPH-2.
          ⋮

PARAGRAPH-2.
```

After the execution of the STRING statement, the data item WISH-LIST contains 'GEEb̸Ib̸WISHb̸Ib̸WASb̸Ab̸FORTRANb̸PROGRAMMER'.

The opposite is achieved by the UNSTRING statement which causes data in a sending field to be separated based on one or more delimiters, and placed into multiple receiving fields. A delimiter may be a single character or a combination of characters.

The SIGN clause allows a user to specify whether a separate character position is used for the sign in numeric items and to indicate whether the position of the sign is leading or trailing.

The PROGRAM COLLATING SEQUENCE clause permits one to specify ASCII, or some other collating sequence. However, changing the collating sequence for a program will necessarily require consideration of all statements which depend upon a character's relative position in a given collating sequence; for example, the SEARCH statement, the SORT/MERGE statement, or alphanumeric comparisons in an IF statement.

Library

Several major changes have been made to the Library module. There are now no restrictions on where a COPY statement may appear in a COBOL program. A COPY sentence may appear anywhere that a COBOL word may appear. As a result, rather strange looking source code can be produced, e.g.,

```
    ADD COPY XX. TO B.
```

If the content of the library text XX contains the single identifier A, the results after the COPY takes place are:

```
    ADD A TO B.
```

There can be more than one library available at compile time. In this case the COPY sentence must contain the name of the library in which the text resides. The presence of the REPLACING phrase in a COPY sentence causes the library text being copied to be edited prior to being inserted in the program. There are several levels of editing that can be used. Words, literals, identifiers and pseudo-text can be replaced by like or different types of operators.

Pseudo-text is bounded by pseudo-text delimiters, which are matching sets of double equal signs (= =), much like a nonnumeric literal which is delimited by sets of matching quotation marks. The replacement of pseudo-text is based on finding a matching set of character-string(s) contained in the library text. The replacement string may be longer or shorter than the characters replaced.

The following COPY statement illustrates the use of pseudo-text to edit a library entry.

User's COBOL Library LIBRARY-TEXT

```
01   ID1 PIC X(54) VALUE IS
     'b̸DEFEATb̸DEDUCTb̸DEFENSEb̸
     DETAILb̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸
     b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸'.
     COBOL Source Statement:
         COPY LIBRARY-TEXT REPLACING
       = =b̸DE= = BY = =THEb̸= =
       = =AT= = BY = =ETb̸OFb̸= =
       = =CT= = BY = =CKb̸GOb̸OVERb̸= =
       = =SE= = BY = =CEb̸BEFOREb̸= =
```

The compiled results of a program with the above COPY statement are the same as a program in which the following source code appeared:

```
01  ID1 PIC X(54) VALUE IS
      "THE␢FEET␢OF␢THE␢DUCK␢GO␢OVER␢
                     THE␢FENCE␢BEFORE␢THE␢TAIL".
```

THE DEBUG MODULE

In the 1968 Standard there were no explicit procedures for specifying what debugging actions, if any, would take place during the execution of a COBOL program. The DEBUG module for the 1974 Standard includes language elements which are designed for debugging COBOL programs. Their inclusion in the COBOL language permits a user to consider the debugging of source programs in the design of an application, not as an afterthought when problems are encountered.

The USE FOR DEBUGGING statement identifies the user items that can be monitored by the associated debugging section. The debugging algorithm which is defined in the debugging section can be controlled by both a compile time switch and an execution time switch. Debug lines are source statements whose inclusion or omission from an object program is controlled by a compile time switch. There is a special register called DEBUG-ITEM which can be accessed in debugging sections. This special register contains information relative to the source code which causes the execution of a debugging section.

INTER-PROGRAM COMMUNICATION

The Inter-Program Communication module provides a means for a program to transfer control to one or more subprograms and the sharing of data among these programs. The action which is taken when there is not enough object time memory can be specified and the memory areas occupied by called programs can be released and made available to the operating system.

Features

The CALL statement causes control to be transferred from one object program to another object program in the same run unit. If the subprogram name is known at compile time, the CALL statement operand is a nonnumeric literal. The subprogram name can also be specified dynamically as the contents of a data-name.

The data items which are shared with the called program are specified in the USING phrase of the CALL statement. In the subprogram, the shared items are specified in the USING phrase of the PROCEDURE DIVISION header and defined as data descriptions in the LINKAGE SECTION. There must be a one-to-one correspondence between the operands in the CALL statement and the operands in the PROCEDURE DIVISION header of the called program. The data descriptions for corresponding operands must define an equal number of character positions but the data descriptions do *not* have to be identical. No space is allocated in the called program for items defined in the LINKAGE SECTION, and references to items in the LINKAGE SECTION are resolved at object time.

The CALL statement also permits the user to specify the action to be taken when there is not enough memory available for a subprogram. This is done in the imperative-statement of the ON OVERFLOW phrase of the CALL statement.

The CANCEL statement releases the memory area occupied by the program referred to in the CANCEL statement. A CALL to a program that has been cancelled causes that program to be loaded and executed in its initial state.

The EXIT PROGRAM statement marks the logical end of a called program. Control is returned to the calling program when this statement is executed. More than one EXIT PROGRAM can appear in a subprogram, but the EXIT PROGRAM statement must be the only sentence in a paragraph.

Inadequacies

The INTER-PROGRAM COMMUNICATION module restricts the possible operands which may appear in the USING phrases. The operands must refer to 77 or 01 level-number items and may not be defined in the REPORT SECTION of the calling program.

There are implementor-defined areas in the Inter-Program Communication module which could cause problems in program portability between systems. The relationship between the operand in the CALL/CANCEL statement and the referenced program is implementor-defined. This means that even though the program-name is a user-defined word and thus could be 30 characters, the implementor may limit the actual number of characters which are used to establish linkage between the called and calling programs. If a user has subprograms on a system which recognizes the first 10 characters of the program-name and moves to an implementation which recognizes the first six, then any referenced program-names with the first six characters identical would be treated as referencing the same subprograms.

The action to be taken when there is not enough memory available for a called program can be specified in the ON OVERFLOW phrase of the CALL statement. If this phrase is not specified, the effects of the CALL statement are defined by the implementor.

THE INPUT-OUTPUT MODULES

The major enhancement of COBOL 74 over COBOL 68 is in the revision to the input/output modules. The

Sequential I-O module has been revised, and the Random Access module has been replaced by two new ones; Relative I-O and Indexed I-O, with some degree of functional and syntactic similarity existing between the new Relative I-O module and the old Random Access module. Taken as a group, the three I-O modules provide the COBOL programmer with the file handling capabilities which are well beyond what has been possible previously.

Sequential I-O

The Sequential I-O module for the 1974 Standard has all the features of the 1968 Standard with the exception of user defined labels and declarative label processing sections, which are no longer supported by the COBOL language. The data-name option of the LABEL RECORDS clause and the USE statement option for label record processing were deleted.

A major new feature in the 1974 Standard allows a user to process character code sets other than the system's native character code set on input and output operations. The CODE-SET clause of the File Description entry specifies the character code set which is used to represent the data on the external media. When this clause is included in a File Description entry, characters are converted to the native character set on input or converted from the native set to the code specified on output. The CODE-SET clause can only be used in File Descriptions which are not mass storage files and the external code representations are limited to the ASCII code set, the native character code set or other character code sets supported by the implementation.

The Sequential I-O module provides the capability to add records to the end of an existing sequential file. Execution of an OPEN EXTEND statement positions a file immediately following the last logical record of the file. Subsequent WRITE statements for that file add records to the end of the file as if the file had been opened in the OUTPUT mode.

The FILE STATUS clause of the File-Control entry and the REWRITE statement are also elements of the Relative I-O and Indexed I-O modules and these two new features are discussed later.

In the Sequential I-O module for the 1974 Standard one can describe the logical page format for a printer-destined file through the LINAGE clause in the File Description entry. The LINAGE clause specifies the size of the top and bottom margins for a logical page, the number of lines comprising the page body, and the line number within the page body where the footing area begins. The values given in the LINAGE clause may be specified as integer constants or the contents of data-names. If the values are integer constants each page has the same format throughout execution of the programs. If the values are the contents of data-names, the values at the time the file is opened specify the first logical page. Each time a new page is started, the values of the data items are examined to determine the values for the current page. Thus the logical format and size of a page can be changed for each new page in a file designated for printer output.

New Input-Output Modules

The INDEXED I-O and RELATIVE I-O are new modules in the 1974 Standard. Because the Indexed I-O module is the most complex and important of the two, we will concentrate our remarks on it, mentioning important features of the Relative I-O module where appropriate.

Features

The INDEXED I-O and RELATIVE I-O modules provide the capabilities for accessing a file in a predefined mode. INDEXED I-O also provides the capability of defining several paths of information retrieval.

In a Relative file, records may be stored "randomly", but are identified by a relative record number on which record storage and retrieval is based. The record number, or key, must be unique, and is the only means by which the file may be accessed. The record storage relationship for an Indexed file however is based upon one or more indexes associated with the file. Thus, an Indexed file may be accessed through one or more record keys.

The prime and alternate keys are defined within the SELECT clause of the FILE-CONTROL paragraph for Indexed files. The contents of the prime key must be unique for each record in the file. This is the base key from which the file is constructed, and is used for inserting, updating, and deleting records. The user may specify one or more alternate keys for the file. Unlike the prime record key, alternate keys may be non-unique.

The content of the keys which are used to retrieve records are considerably more flexible in the Indexed I-O module than in the Relative I-O module. Under the Relative I-O file organization the record reference key must be an unsigned integer and is defined outside the record description entries for the file. In the Indexed I-O organization the record reference key must be an alphanumeric data item which can contain any combination of characters in the computer's character set, and must be defined within a record description entry for the file.

The language specification for COBOL 74 allows for sequential, random or dynamic access within both Relative I-O and Indexed I-O modules. The access mode for a given file is indicated by use of the ACCESS MODE IS RANDOM or ACCESS MODE IS SEQUENTIAL clause. When the ACCESS MODE IS DYNAMIC is specified, records may be processed either randomly or sequentially through use of the appropriate I-O statement. The access mode for a file need not be the same as the mode in which the file was created. When accessing the file sequentially, the records are retrieved in ascending order based on the key contents. The START statement provides positioning within an indexed or relative file for subsequent sequential retrieval of records. When processing the file randomly, records are stored or retrieved based on the data contents of the record key. Records are accessed based on the current key of reference. The key of reference, prime record key or alternate record key, is established at the COBOL instruction level. The default key of reference is the prime record key, but an alternate key of reference may be specified in a random READ statement. Any subsequent sequential read uses the key of reference established by the last random read or START statement.

File maintenance

The means of maintaining mass-storage files, i.e., record insertion, record deletion and record updating is accomplished through the use of the verbs DELETE, WRITE and REWRITE. Only the prime record key associated with the file is used in providing all file maintenance functions, i.e., RELATIVE KEY for Relative I-O and RECORD KEY for Indexed I-O.

The DELETE verb logically removes a record from the file. Once the statement has been executed, the record cannot be accessed again. The WRITE and REWRITE verbs insert and update, respectively, the records in the file. Any file maintenance key associated with the file must be unique within all the records for the file. The previous input-output statement for the file must have been a READ statement. The REWRITE statement causes the last record READ by the program to be logically replaced by the specified record. The number of character positions in the record being rewritten and the record being replaced must be equal. The WRITE statement causes a record to be inserted assuming the key does not already exist.

An important addition to the 1974 language specification for all I-O modules is the FILE STATUS data item which contains information as to the success (or failure) of an I-O operation. The FILE STATUS clause, located in the FILE-CONTROL entry for a file, specifies a two character alphanumeric data item and contains values which indicate the results of every statement which references that file explicitly or implicitly. The operating system moves the values into the file status data item upon the completion of any statement which references the file.

The new I-O modules provide the user greater flexibility and a wider range of functions than have been previously available. The Indexed I-O module in particular gives the user the ability to implement multi-key retrieval functions and provide a closer relationship of the capabilities of data base management systems entirely within the scope of the COBOL Language.

THE COMMUNICATION MODULE

The motivating factor behind including a "Communication Module" in COBOL 1974 was the advent within the past decade of computer systems using remote terminals and the use of these terminals for message processing applications. Heretofore, the COBOL user has been unable to perform this class of interactive operations without resorting to system dependent facilities such as assembly language support routines. Many implementations of COBOL permitted limited degree of access to remote terminals via the ACCEPT and DISPLAY verbs, but obviously these posed serious limitations on capabilities by limiting transmission to an "on demand" basis and in no case could a program be notified of unsolicited input from a terminal.

The Communication module in COBOL 1974 attempts to solve these deficiencies by providing four new I/O verbs (RECEIVE, SEND, ENABLE, DISABLE) and interfacing COBOL programs to any configuration of remote terminals via a set of message queues. The operation of the message queues and the remote terminals is handled by a Message Control System (MCS), a "black box" software package which is largely implementor defined and by its very nature is system-dependent. The MCS must provide the logical interface between the COBOL communication object program and the systems network of communication devices by performing line discipline, including such tasks as dial-up, polling, and synchronization, and by performing device-dependent tasks such as character translation and insertion of control characters. The COBOL programs interface with the MCS through the programs' Communication Descriptions or CD's. CD's are placed in the Communication Section which follows the File, Working-Storage, and Linkage Sections in the Data Division. The CD establishes either an input or an output path for messages, and provides parameter fields for passing information between the program and the MCS.

Features

For the Procedure Division, the Communication Module introduces four new verbs; RECEIVE, SEND, ENABLE, and DISABLE; plus a new variation for an old one, ACCEPT. The following paragraphs summarize these statements as presented in the formal language specification for the Communication module.

"The RECEIVE statement makes available to the COBOL program, a message, message segment, or a portion of a message or segment." Prior to executing a RECEIVE the user must specify in his input CD record area, the name of the queue or subqueue he wishes to address. Executing the RECEIVE then effects dequeuing of a message from the appropriate queue and placing of that message into the field designated by the user. If the addressed queue is empty, then at the user's option, the program can be forced to wait until a message becomes available, or it can be directed to proceed immediately with execution of the next sequential statement. During the RECEIVE operation, all data items in the input CD record are updated by the MCS.

"The SEND statement causes a message, a message segment, or a portion of a message or segment to be released to one or more output queues maintained by the MCS."

Prior to executing a SEND, the user must specify in his output CD record area, the number and names of the destinations to which the data is to be sent, plus the length of the text in characters. In the SEND statement itself, the user specifies which transmission sentinel is to be used, end-of-message, end-of-group, or end-of-segment. When a SEND is executed, the MCS must enqueue the data on the appropriate queue(s) and return the operation status to the CD for use by the program. Line and page control may be exercised on line oriented devices.

A variation of the ACCEPT statement enables the user to determine the number of messages currently enqueued in any particular queue. Prior to executing the ACCEPT statement, the user must specify in his input CD, the name of the queue or subqueue whose size is to be returned. The MCS will return to the CD in the appropriate parameter fields, the message count and the status of the operation. Only input queues may be measured in this way.

The ENABLE and DISABLE statements direct the MCS to allow and inhibit respectively, data transfer between specified output queues and destinations for output, or between specified sources and input queues for input. The queues or destinations involved must be named in the appropriate CD before execution of the statement. These statements make and break the logical connections between the queues managed by the MCS and the network of communication devices, but they do not affect the logical connections between the various queues and the program itself. The specification of a key or system password is required in both statements "in order to prevent indiscriminate use of the facility by a COBOL user who is not aware of the total network environment, and who may therefore disrupt system functions by the untimely issuance of ENABLE and DISABLE statements".

A special option permits the user to specify the symbolic name of a specific device in his input CD, and then request enabling or disabling of the logical paths between that device and *all* queues and subqueues linked to it.

Finally, there is the added capability to designate in a COBOL communication program that it is to be scheduled for execution automatically by the MCS whenever the MCS determines that there is message processing to be done. This is accomplished by specifying an option in one input CD in the program. Subsequently, when the MCS invokes the program, it will place the name of the queue or subqueue, which prompted the action, in the appropriate parameter fields of the input CD. There are means to test within the program whether the program was invoked by the MCS or scheduled through job control language.

Inadequacies

The basic problem with the Communication module as it now stands is the fact that the MCS and its interface to the network of peripheral devices is so ill-defined. Although the concept of the MCS is never formally introduced except in Appendix C of the Standard, its existence is implied throughout the specification for the Communication module by frequent references to it (the MCS) by name. Unfortunately, the appendices are not considered a part of the formal COBOL language specification; and thus they are in no way binding. At best, the appendix can be considered a suggested guideline for implementation. In other words, it can only serve to enlighten the reader as to what the Programming Language Committee (PLC) of the Conference on Data Systems Language (CODASYL) might have had in mind when they designed the specification for the Communication module.

Contributions

It is not really possible to assess the usefulness of the new Communication module. Such an assessment must await the use of the features in a communication environment. The features themselves are not extensive, and the heart of the system, the MCS, is not well defined. These considerations suggest a degree of skepticism vis-a-vis the degree of applicability of the Communication module. Despite this skepticism, it must be admitted that some capability is now available for message handling.

RECOMMENDATION

COBOL 74 is in many ways a vast improvement over COBOL 68. New features have been added which contribute to the capability of the language (the new I-O modules), enlarge its scope (the Communication Module), and enable the programmer to produce a better product (the Inter-program Communication and Debug modules). Furthermore, old modules have been enlarged and improved. We believe the efforts of the standardizing body, ANSC X3J4 (COBOL) should now be directed toward producing a more complete and precise definition of the language.

The procedures of the American National Standards Institute require that action be taken to reaffirm, review, or withdraw the standard no later than five years following the publication of the current standard. X3J4 should seriously consider a radical rewrite of the standard in an effort to make it more a standard and less a generalized user's manual. A "standard" is supposed to be an authoritative measure by which correctness of other things may be determined (condensed from Websters'), but it is a difficult task to measure correctness against a standard which is ambiguous and subject to interpretation. Both the syntax and semantics of the language should be expressed as formal grammars and defined in an appropriate metalanguage. The efforts of ANSC X3J1 to so define the PL/1 language is an admirable example of what can be done. When a language is well defined, there can be no ambiguities and no doubt as to what is valid and what is not. There is seldom a need for interpretation. Such definitions are also of significant value when a compiler implementor chooses to use a syntax-directed or other automated parsing technique. Finally, a precise definition would make much simpler the task of determining the degree to which COBOL compilers conform, in their translation of COBOL programs, to the Standard.

REFERENCES

1. Baird, G. N., "The DOD Compiler Validation System," *Proc. 1972 FJCC*, AFIPS Press, Volume 41, pp. 819-827.
2. *American National Standard Programming Language COBOL*, X3.23-1974, American National Standards Institute Incorporated, New York 1974.
3. *American National Standard COBOL X3.23—1968*. American National Standard Institute Incorporated, New York 1968.
4. *Federal Information Processing Standards*, Publication 21-1, U. S. Government Printing Office, Washington, D. C., (pending).

COBOL '74—Contributions to structured programming

by PAUL OLIVER
Department of the Navy
Washington, D.C.

INTRODUCTION AND SCOPE

Gerald Weinberg, in his book *The Psychology of Computer Programming*[1] suggests that we cannot really measure the goodness of programs on an absolute scale, and that we generally cannot even measure them on a relative scale. There is indeed evidence that rapid quantification of software quality is not really feasible,[2] because simple formulas can often be misleading and hence not very credible. One could, for example, measure program complexity in terms of the fraction of program statements which are branch statements. Consider however two programs, one with 10 decision points leading to 10 different tasks, each consisting of one instruction, and one with 3 decision points leading to 3 tasks, each consisting of 30 instructions. The "complexity" rating of the first would be 0.50, while that of the second would be 0.09. Surely two such programs would not differ so greatly in complexity.

Thus, we are led to less quantifiable measures of software quality, and a cursory glance at recent literature indicates that "simplicity" is a most desirable characteristic. The simplification of a complex task can be achieved by modularizing it into separate, smaller tasks. We further require that each task be discrete and visible, that it be self-contained (thus constraining the assumptions it makes regarding the implementation of other tasks), that it have a single entry point and a single exit, and that when invoked by another task or module it returns to a standard point.[3]

One way of enhancing the simplicity of a program is through structured modularity, which is achieved through the separation, within a module, of data, processing code, and control, and through the maintenance of a simple, visible, control structure.

This paper concerns itself with how this can be accomplished using the COBOL language as defined in the 1974 COBOL Standard.[4] We will look at how structured constructs can be simulated using standard COBOL verbs, and discuss the micro-efficiency of using these constructs versus less disciplined techniques. We also will point out certain deficiencies in COBOL 74 which can impair program clarity, and make suggestions on future improvements.

IMPLEMENTATION OF STRUCTURED CONSTRUCTS

Structured programming may be viewed as a set of rules designed to enhance a program's readability, thereby reducing stylistic differences between programs written by different individuals, and improving a programmer's ability to understand and modify existing programs. The rules are, by now, well known:

(a) Use of code formatting conventions (e.g., indentation) to typographically represent control logic.
(b) Limit subroutine size (e.g., to the number of lines that can fit on a listing page).
(c) Have but a single entry and a single exit per subprogram.
(d) Limit the control forms to three basic structures,

(S = statement, P = predicate)
○ Sequence: S_1S_2.
○ If-then-else: If P then S_1 else S_2.
○ Do-while: While P do S.

The basic tool in COBOL for implementing structured programming is the PERFORM verb. Experience has indicated that it is wise to use the THRU option to define the scope of the PERFORMed code, since there is a tendency to neglect definition of the scope of a SECTION. In the following discussion, some familiarity with structured programming and COBOL is assumed.

If-then-else

The if-then-else construct can be implemented in three different ways. The most common is

```
IF     condition
       statement-1
       statement-2
         . . .
       statement-n
```

```
ELSE
      statement-n+1
      statement-n+2
         . . .
      statement-n+m.
```

If the condition is "true" control passes to statement-1, and at the completion of statement-n, control is given to the statement following statement-n+m. Note that COBOL does not explicitly express the "thru" (perhaps it should be included as a noise word in future specifications).

If n and m are large, a clearer form of the construct is

```
IF    condition
      PERFORM paragraph-1 THRU paragraph-1-exit
ELSE
      PERFORM paragraph-2 THRU paragraph-2-exit.
```

An additional variation must be introduced if either the IF or ELSE clause include a nested IF, followed by one or more statements which must be executed regardless of the result of the nested IF. In such cases, the nested IF must be PERFORMed:

```
IF    condition-1
      PERFORM nested-if-test
      statement-2
ELSE
      statement-3
         . . .
nested-if-test.
      IF condition-2
        statement-1
           . . .
```

In this example, statement-2 should be executed without regard to the evaluation of the inner IF test. This particular variation could be avoided if COBOL had a DO . . . END type of "vertical parentheses" as in PL/1, which allow the statements between the DO and the END to be treated as a single syntactical unit by the compiler.

Do-while

The do-while construct must be "simulated" in COBOL by the use of PERFORM verb with the UNTIL option. The UNTIL option is similar to the WHILE construct in that testing of the condition predicate occurs prior to any code being executed, but differs from WHILE in that termination occurs when the condition is "true" rather than "false." Thus, the do-while construct is:

```
PERFORM paragraph-name THRU paragraph-name-
  exit UNTIL (NOT P).
```

The negation of P is necessitated by the aforementioned difference in the termination condition between UNTIL and WHILE.

Indexed looping is implemented by:

```
PERFORM paragraph-name THRU paragraph-name-exit
  VARYING iden-1 FROM iden-2
  BY iden-3
  UNTIL (NOT P).
```

Do-until and case

Two additional constructs can be included for the sake of convenience. These are do-until and case.

Do-until is essentially an iterative DO-loop. It differs from do-while in two ways:

(a) The predicate is tested *after* each execution of a statement sequence, not before, so that the statement sequence is executed at least once.

(b) The test on the predicate is reversed, so that the do-until terminates when the predicate is "true".

In COBOL, do-until is implemented by:

```
PERFORM paragraph-name THRU paragraph-name-exit.
PERFORM paragraph-name THRU paragraph-name-exit
  UNTIL P.
```

Note that the first execution of the PERFORMed paragraph must be forced artificially.

The case construct is a multi-way switch, and is implemented in COBOL by using PERFORM, and GO TO . . . DEPENDING ON, i.e.,

```
PERFORM case-paragraph THRU case-end.
  . . .
  . . .
case-paragraph.
  GO TO case-1, . . . , case-n
    DEPENDING ON integer.
  GO TO case-end.
case-1.
  (code for case-1)
  GO TO case-end.
  . . .
case-n.
  (code for case-n)
  GO TO case-end.
case-end.
  EXIT.
```

The multi-way switch may also be implemented using the ALTER statement:

```
IF input-type=1
  ALTER case-edit to case-1.
. . .
IF input-type=N
  ALTER case-edit to case-n.
case-edit.
  GO TO case-i-fix.
. . .
```

This implementation is *not* recommended, for reasons which will be discussed in a later section.

Micro-efficiency of the structured constructs

Many COBOL users have expressed the concern that the extensive use of structured constructs (i.e., avoiding the GO TO) may create inefficiencies in terms of execution time and memory utilization. Our experience indicates that some degradation may indeed occur, but its degree is slight.

Consider for example the use of PERFORM ... THRU versus GO TO. On the IBM S/360 V4 COBOL Compiler a GO TO translates to a conditional branch. Thus, execution of a paragraph requires 2 instructions (one to go, one to return), there is less flexibility (the paragraph cannot be "executed through"), and the control structure is hidden. Execution of PERFORM ... THRU requires five instructions, the PERFORMed paragraph can be executed in-line, and the control structure is clear.

Looping is accomplished by using the do-until construct. This requires 18 machine instructions for one compiler we have examined. Consider however the alternative (by using GO TO). This would require the programmer to:

- Set lower and upper bounds, and the increment.
- Increment the index.
- Test the condition and branch.
- Reset the index.

These tasks require 14 machine instructions on the same compiler. The resultant saving of 4 machine instructions hardly warrants the loss of control logic visibility.

Finally, consider the use of GO TO ... DEPENDING ON versus ALTER as a means of implementing the case construct. A 4-case example required 15 instructions on an HIS 6050 using the GO TO ... DEPENDING ON instruction. A series of 4 IF ... ALTER ... statements required 32 instructions. Not only does ALTER obscure a program's control structure, but in fact is not a very efficient verb.

We have found that attempts to recode existing programs so as to delete GO TO's can result in confusion. The use of structured programming as a discipline must be approached at the level of program formulation if it is to be successful. The Federal COBOL Compiler Testing Service recently recoded the executive program for the COBOL Compiler Validation System.[5] This program is about 3000 COBOL instructions in size and is quite complex. By reformulating the system we were able to recode it at the cost of some 5 compilations. Of three severe errors that we encountered, 2 were caused by GO TO's which had been retained in the program.

The inter-program communication module

Modularity is a byproduct of structured programming. Modularity in large programming systems is difficult to achieve without a true subroutine capability. This capability is provided in COBOL 74 through the interprogram communication (IPC) module. It allows the transfer of control from one program to another within the same run unit (through the CALL statement) and for sharing of data, or parameter passing (through the USING option). Additionally, programs whose name is *not* known at compile time may be invoked at execute time, and the availability of memory for a called program may be determined at execute time. While these features have been present in some COBOL implementations, they now have been placed within the standard for the first time. This is surely one instance in which COBOL 74 is a major improvement over COBOL 68. This capability not only enhances modularity, but allows for truly reusable code by providing the means for easily detaching and reattaching subprograms. The presence of the LINKAGE section allows for the grouping of passed parameters, thereby providing good interface visibility as well as some "locality" of data.

DEFICIENCIES AND FUTURE REQUIREMENTS

The revised COBOL language possesses many features which make a positive contribution to the writing of clear, simple, and therefore (hopefully) reliable programs. Regrettably, the language also has many deficiencies which tend to inhibit good programming.

Perhaps the most serious of these is the sheer size of the language. Ideally, a programmer using COBOL for a particular class of applications should have to understand only those features of the language pertinent to that application; while the programmer using COBOL in a broader way should be able to use and understand all of its features. This requires modularity of design, but also smooth integration of the modules. Yet, someone wishing to use the RELATIVE I/O module of the current standard (Reference 4) needs to know many of the details of the NUCLEUS, SEQUENTIAL I/O, and TABLE HANDLING. It should be noted in passing that the 1974 Standard document is over 500 pages long. Jean Sammet required but 16 pages to describe the basic elements of COBOL 61 (Reference 6). One wonders if the two documents are really describing the same language. Even a simple verb such as DIVIDE has five formats in COBOL 74, with a minimal compiler implementation requiring three of them. Further aggravating the feasibility of a user knowing only part of the language is the large number of reserved words (keywords) in COBOL 74—over 300 of them!

C. A. R. Hoare[7] cites simplicity of design as a necessary condition if a language is to assist the programmer in program design, documentation, and debugging. The designers of COBOL appear to have replaced the objective of simplicity with that of modularity, but as the above example suggests, the result has been a "modular" language which is nevertheless quite complex.

Lack of rigor in the COBOL language specification (i.e., Reference 4) is a particularly troublesome shortcoming. An instance of this is the large number of "undefined" results. The specifications for the ENABLE verb in COBOL 74, for example, are not explicit as to what is to occur if an ENABLE is issued with respect to a device that is already

ENABLEd. Clearly *something* must take place. This means that the language specified by a given COBOL implementation will differ from that specified by the standard. This same problem is also caused by the many instances in COBOL where the specifications are "implementor defined," as is the case with the COMPUTE verb.

It has been pointed out in an earlier section that a form of "structured" programming is possible using COBOL. This does however require a "simulation" of the structured constructs (as in the do-until case). Use of the substitutes may indeed enhance control logic visibility, but they are more awkward to use than would be the unencumbered constructs. (It should be noted that there are several commercially available "pre-compilers" which allow the use of the simple structured constructs.) Furthermore, complete "structuredness" is precluded by a lack of vertical parentheses (or block structure) and an absence of "local" data. The former shortcoming can be particularly annoying where IF . . . THEN . . . ELSE cases are nested.

Finally, there are features in COBOL 74 whose use tends to obscure a program's control structure beyond normal bounds of good taste. The possibility of asynchronous exits (e.g., the AT END clause of the READ verb) will obviously alter the control sequence of a program. This "flaw" is not, of course, peculiar to COBOL. One way of alleviating the problem is to test for a "flag" in the program control flow, and to set the flag upon the occurrence of an asynchronous interrupt.

Consider, for example, use of the ADD statement with the ON SIZE ERROR option. If the ADD takes place in the control sequence of statements, the usual GO TO upon an occurrence of the sum interrupt would disrupt this sequence. One could however PERFORM the code which includes the ADD statement. The ON SIZE ERROR option would be used to set a flag upon the occurrence of an overflow. The statement following the PERFORM could then test the flag for a given value, and perform the test appropriate to that value. This would of course be done using the if-then-else construct.

The ALTER verb is a particularly offensive weapon in the hands of anyone wishing to obscure a program's control logic. Briefly, the ALTER statement causes the modification of the destination paragraph of a given GO TO statement (see Reference 4 for a complete and precise definition). The very fact that the ALTER verb modifies a predetermined sequence of operations inhibits good programming practice. The control sequence may furthermore be additionally obscured by the fact that GO TO statements which have been modified by an ALTER statement, and which are contained in independent program segments may, in some instances, be returned to their initial states.

CONCLUSION

The modules which have been added to COBOL 68 resulting in COBOL 74 have increased the language's generality to a level which is at least comparable to PL/1. One may argue that it still falls short of true generality, since its applicability in areas such as simulation, list structure processing, graphics and even numerical computations is less than self-evident. Still, it is as "general" as one may reasonably expect at this time. This trend away from the purely commercial applicability is welcomed. The need to have easily transportable programs is increasingly pervasive, and is more easily accomplished in an environment where one general purpose language is used. Furthermore, today's programming problems seldom can be neatly classified under a single application field, and this is certainly true in "commercial" environments.

Included in this expanded language are several features which encourage good programming practices. One hopes that further improvements will be made, and that those features detrimental to good programming will be deleted from COBOL.

REFERENCES

1. Weinberg, G. M., *The Psychology of Computer Programming*, Van Norstrand Reinhold, 1971.
2. Boehm, B. W., et al., *Characteristics of Software Quality*, TRW-SS-73-09, December, 1973.
3. Armstrong, R. M., *Modular Programming in COBOL*, John Wiley and Sons, 1973.
4. *American National Standard Programming Language COBOL*, X3.23-1974, American National Standards Institute, Inc., 1974.
5. Baird, G. N., "The DOD COBOL Compiler Validation System," *Proceedings of the Fall Joint Computer Conference*, 1972.
6. Sammet, J. E., "Basic Elements of COBOL 61," *Communications of the ACM*, Volume 5, No. 5, May 1962, pp. 237-253.
7. Hoare, C. A. R., *Hints on Programming Language Design*, Stanford Artificial Intelligence Laboratory, Memo AIM 224, STAN-CS-73-403, December, 1973.

Program debugging using COBOL '74

by GEORGE N. BAIRD
Department of the Navy
Washington, D.C.

INTRODUCTION

The testing and checkout of production software often consumes upwards of 50 percent of the effort that goes into the development of the system.[1] The absence in most programming languages of language elements specifically defined for debugging programs contributes to the time and effort involved in the checkout of systems due to programmers having to use various techniques to improvise debugging code. This was especially true of the 1968 COBOL Standard[2] in that there were no language elements dedicated to the debugging and proving correctness of programs.

The purpose of this paper is to explore the concepts and language constructs introduced into the 1974 COBOL Standard[3] as the Debug Module. What this module has done for the revised COBOL Standard is to provide a means by which the user can describe his debugging algorithm at the source level. This includes the conditions under which data items and procedures are to be monitored during execution of an object program.

For the purposes of this paper, implementor provided techniques will be for the most part ignored. This is due to the wide number of verbs and the various techniques which have emerged as a result of a lack of any formal definition for debugging source statements. The specifications presented in X3.23-1974[3] should provide in a form of standard syntax a semantic composite of implementor debugging techniques. The new debugging module permits the user to determine what to monitor and what information should be provided to the programmer for the actual debugging of the program.

COBOL DEBUGGING WITH X3.23-1968 (AMERICAN NATIONAL STANDARD COBOL)

Prior to the revised COBOL Standard, there were no explicit procedures for specifying what debugging, if any, would take place during the execution of a COBOL program. Although not designated as such, there are language elements in X3.23-1968 that can be used for debugging. The DISPLAY statement, for example, causes low volume data to be transferred to a specific output device. By strategically placing DISPLAY statements in a source program the user could attempt to follow the logic of the program as execution took place. This could be accomplished by DISPLAYing a procedure-name, or a data-name and its contents.

```
PROCEDURE-NAME-1.
     DISPLAY  "PROCEDURE-NAME-1" UPON
     PRINTER.
     NOTE: THE ABOVE STATEMENT INDICATES
     THAT THIS PROCEDURE WAS EXECUTED.
     MOVE 000000 TO COUNT-B
     ADD 1 TO COUNT-B.
     DISPLAY "COUNT-B=" COUNT-B UPON
     PRINTER.
```

The above sequence of statements would show, in the form of output to the printer, that PROCEDURE-NAME-1 was executed each time control was passed to that set of procedures. A few statements later the contents of the data-item COUNT-B would be provided. Execution of the DISPLAY statements would result in the following printed output:

```
   :
   PROCEDURE-NAME-1
   COUNT-B=000001
   :
```

This method of tracing a program's execution is crude at best, and prone to error. For example, once a program has been debugged and is ready to be placed in a production mode, one hardly wants either the debugging information provided or the overhead in the form of the object code generated due to the presence of the debugging statements. Therefore the source code used for debugging the program must be removed. This change in the source program could result in errors being introduced into the program (either syntax or logic) which could necessitate additional testing.

Another consideration in the discarding of debugging statements is the maintenance which might be required during the life of the program. If the requirements for the program change, the source program will have to be modified and the need for debugging and testing arises again. The reintroduction of the source statements necessary to accomplish the debugging of the program once again could introduce syntax and/or logic errors. One solution to the problem is to control the execution of the source code for debugging sessions by the use of conditional statements associated with each debugging statement:

```
IF DEBUG-REQUIREMENT EQUAL TO "YES"
  DISPLAY "COUNT-B=" COUNT-B UPON
  PRINTER.
```

The debugging code thus could be executed when needed, and once the program was ready for production the execution of the debugging code could be suppressed. (The control could be in the form of a switch setting, parameter card, value in the Working-Storage.) However, the overhead problems would still exist; the larger program size due to the debugging object code being present, and additional execution time required for testing the debugging requests.

IMPLEMENTOR EXTENSIONS FOR DEBUGGING

The need for debugging functions was satisfied partially by implementors providing language extensions, in their compilers, which were designed for use in debugging the programs, (i.e., TRACE, EXHIBIT, MONITOR statements etc.). The results were inconsistent among implementations as to the name of the verb used and the amount of external control that was available relative to the execution of the debugging statements. These inconsistencies in control included (1) whether the debugging code had to be removed for production runs or the execution of the code could be controlled at execution time, as well as (2) whether the generation of object code could be suppressed if the debugging statements were not removed. The innovative ability of the implementor was certainly the determining factor in this situation as to whether the implementation was an "intelligent" implementation or something with less sophistication.

COBOL DEBUGGING BASED ON X3.23-1974, THE REVISED COBOL STANDARD

The revision to X3.23-1968 has, for the most part, solved the above shortcomings by more realistically providing the tools in COBOL for accomplishing debugging in a reasonable fashion.

The ideal solution, it appears, would be the ability to control, in a way external to the object program, whether the debugging code would be executed. Also, the ability of having the debugging source code remain in the source program and only be compiled when requested would eliminate the problem of source code changes merely for debugging purposes.

A brief description of the capabilities of the debugging module follows:

1. A compile time switch determines whether the debugging source code will be used to generate object code.
2 An object time switch determines, (if the compile time switch caused the debugging statements to be compiled) whether the debugging object code will be executed.
3. Debugging lines are source statements that take on either the characteristics of debugging statements, or comment lines depending on the setting of the compile time switch.
4. The USE FOR DEBUGGING statement specifies the user defined items that are to be monitored by the associated debugging section. The procedural statements in the debugging section define the algorithm by which debugging will take place.
5. A special register called DEBUG-ITEM can be accessed in the debugging sections. This special register contains the source program sequence number of the line in the source program that triggered the execution of the debugging section, the name of the data or procedure or file name involved, and other information relative to the referenced item.

DEBUG MODULE FUNCTIONAL CAPABILITIES

The language elements in the Debug Module of the revised national COBOL standard provide a means by which the user may describe his algorithm to suit his needs. This includes controlling the conditions under which data items or procedures are to be monitored during the execution of the object program. The decision of what to monitor and what information to capture are explicitly in the domain of the user. The introduction of the debug facility merely provides a convenient access to the information pertinent to debugging.

The compile time switch

The capability exists to control, from within the source program, whether debugging source statements are to be compiled or treated simply as comments. The WITH DEBUGGING MODE clause is written as part of the SOURCE-COMPUTER paragraph and serves as the compile time switch over the debugging statements:

```
SOURCE-COMPUTER. Computer-name
  [WITH DEBUGGING MODE].
```

When the optional WITH DEBUGGING MODE clause is specified in a source program, all debugging sections and debugging lines are compiled as regular source statements. The absence of the WITH DEBUGGING MODE clause causes all debugging lines and debugging sections to be compiled as though they were comment lines.

The compile time switch permits all debugging source code to remain intact in the source program and only be compiled when requested. This satisfies the option of being able to retain all debugging source code in the program for future testing or debugging without having to suffer the overhead that would result from merely suppressing the execution of the object code produced, as discussed earlier.

The object time switch

The object time switch dynamically activates or deactivates the debugging code inserted by the compiler based on the debugging sections described in the declarative portion of the Procedure Division. The accessing of this switch, although dynamic, cannot be accomplished from within the program, i.e., set by a COBOL statement, but is controlled outside the COBOL environment. This will most likely be accomplished through the use of a parameter specified in the operating system control language.

If the switch is 'on' then execution of the program includes the execution of all debugging sections and as a result any output generated by the debugging algorithm specified in the program. The switch can activate the debugging code only if the compile time switch was 'on' (WITH DEBUGGING MODE clause specified in the Source-Computer paragraph) when the program was compiled. Had the compile time switch not been 'on' then no debugging code would have been generated and therefore could not be activated. The ability to control the activation of the debugging code at object time permits the running of the program (if desired) with the debugging code present but dormant. If a problem occurs or spot checking is needed, the program could be run with the object switch 'on' and debugging could continue.

Debugging lines

A debugging line is any line with a 'D' in the indicator area (column 7) of a source line. Depending on the presence or absence of the WITH DEBUGGING MODE clause in the Source-Computer paragraph, the line would be compiled either as part of the source program or as a comment line. (A comment line is defined as any line with an '*' in the indicator area and is ignored by the compiler except for presenting it as part of the source listing.) It should be noted that the object time switch has no effect on debugging lines.

The contents of a debugging line must be such that a syntactically correct program is formed with or without the debugging lines being considered as comment lines. Successive debugging lines are permitted and a statement can be continued on successive debugging lines as long as it results in a syntactically correct source program at compile time. Due to the requirement of the 'D' being present in the indicator area, character-strings may not be broken across two or more debugging lines.

Debugging lines can appear anywhere in a source program after the Source-Computer paragraph. An example of debugging lines follows:

```
      COL 7
        ↓
001400   PROCEDURE-NAME-1.
001500D          DISPLAY "PROCEDURE-NAME-1"
001510D          UPON PRINTER.
001600           MOVE 000000 TO COUNT-B.
001700           ADD 1 TO COUNT-B.
001800D          DISPLAY "COUNT-B=" COUNT-B
001810D          UPON PRINTER.
```

This is the same debugging example illustrated previously for the 1968 COBOL Standard using the DISPLAY statement, but here control is achieved over whether the coding will be considered for compilation (compile time switch).

Since debug lines can appear in both the Environment and Data Division, as well as the Procedure Division, not only procedural statements can be associated with debugging, but data and files as well. For example if the output from the debugging session were to be sent to a temporary file and examined later for some reason, the file itself could be described in such a manner that it would only be compiled into the program when the compile time switch was on.

```
002000   ENVIRONMENT DIVISION.
                  :
003000   INPUT-OUTPUT SECTION.
003100D          SELECT DEBUG-FILE ASSIGN TO
                 TAPE.
003200           SELECT PAYROLL-FILE ASSIGN
                 TO DISK.
                  :
004000   DATA DIVISION
004100   FILE SECTION
004200D  FD DEBUG-FILE LABEL RECORDS
004210D  OMITTED.
004300D  Ø1 DEBUG-REC PIC X(200).
004400   FD PAYROLL-FILE
004500       LABEL RECORDS STANDARD.
                  :
005000   PROCEDURE DIVISION
005100   INITIALIZATION SECTION
005200   HOUSE-KEEPING
005300D          OPEN OUTPUT DEBUG-FILE.
005400           OPEN INPUT PAYROLL-FILE.
                 :
006000D          WRITE DEBUG-REC.
                 :
007000D          CLOSE DEBUG-FILE.
```

In the above example a file which is to contain debugging information, and all of the references to it are defined as D lines, and their presence or absence is controlled through the compile time switch.

The USE FOR DEBUGGING statement

For tracing procedure-names and the activity of data items, the debug line would be woefully inadequate or at best pedestrian in nature. The extensive tracing of procedures and contents of data items is accomplished through the use of debugging sections.

The USE FOR DEBUGGING statement is a declarative statement and defines a debugging section. This permits the programmer to establish a debugging section and to identify items that are to be monitored by the associated debugging section. The rest of the declarative procedures in the section define the algorithm by which debugging will take place:

```
DECLARATIVES
  DEB-1 SECTION. USE FOR DEBUGGING ON
                 ALL PROCEDURES.
  PARAG-1.
                 .
                 .
                 .
  DEB-2 SECTION. USE FOR DEBUGGING ON
                 DATA-NAME-1.
                 .
                 .
                 .
  END DECLARATIVES.
```

A debugging section is executed each time, as appropriate, that the named item(s) (whether named implicitly or explicitly) are referenced elsewhere in the Procedure Division. A debugging section allows the user to provide a set of procedures for acting on the data provided in DEBUG-ITEM. The contents of the DEBUG-ITEM are updated by the debug system each time a reference is made to an item for which debugging has been requested.

The COBOL debugging statements themselves produce no debugging output. The debugging section permits the user to analyze what is taking place and selectively take any necessary action which may include producing printed output. This is opposed to the 'shotgun' approach of producing all data provided by the debugging system. It also provides an advantage over the TRACE, MONITOR, EXHIBIT, . . . statements in that not only is more information available, but it is provided to the user in the debugging section rather than being sent to a printer-destined file. The user has the ultimate decision as to what, if anything, is to be printed. The TRACE statement could be simulated very simply:

```
  DEB-1 SECTION. USE FOR DEBUGGING ALL
                 PROCEDURES.
  DEB-PARA-1.
      DISPLAY DEBUG-ITEM.
```

The reference to the special register 'DEBUG-ITEM' is the method of accessing the data that is provided to the user by the debug system. The contents of this data item will be covered fully later, it is sufficient to note that one of the fields contains the name of the procedure that was just referenced/entered/altered.

There are basically four options or types of references that can act as triggers in a debugging section. The following discussion shows the various forms of the USE FOR DEBUGGING statement that can be used and the effect of using each.

The first option of the USE FOR DEBUGGING statement allows for the tracing of the execution of the various sets of procedures in the procedure division (paragraph trace):

$$\text{USE FOR DEBUGGING ON} \begin{Bmatrix}\text{ALL PROCEDURES}\\ \text{procedure-name-1} \ldots\end{Bmatrix}$$

There are two methods of specifying the procedures on which to trap, as is shown in the above USE statement. When the procedure-name-1 phrase is chosen, the debugging section is executed immediately before each execution of the named procedure and immediately after the execution of any ALTER statement which references procedure-name-1. In this case, each procedure-name on which debugging will take place must be specified in a USE statement. (A USE statement may contain a reference to more than one procedure-name.) On the other hand, if the ALL PROCEDURES phrase is specified, then the above described action takes place for each procedure-name in the program; the only exception would be procedure-names specified in debugging sections.

The second option of the USE FOR DEBUGGING STATEMENT is for monitoring the activity of data items referenced in the Procedure Division. This includes monitoring a data item either when its contents have been modified or each time it is referenced. (For the purposes of this discussion the term "identifier" represents a data-name in a COBOL program including all qualifiers necessary to make it unique and all subscripts/indexes required to reference it.)

```
USE FOR DEBUGGING ON
  [ALL REFERENCES OF] identifier-1.
```

If the optional ALL REFERENCES phrase is specified, the debugging section is executed for every statement that explicitly or implicitly references identifier-1. The absence of the ALL REFERENCES phrase causes the debugging section to be executed immediately after the execution of any COBOL statement that references identifier-1 and causes the contents of the data item referenced by identifier-1 to be changed.

There are a few isolated cases in which an implicit reference to an identifier can cause a debugging section to be executed. A case in point is the WRITE statement where the FROM phrase is used:

```
WRITE record-name FROM identifier
```

results in implicit reference to identifier in that it is moved to record-name prior to the record actually being written as though a MOVE statement had been used. This results in an implicit reference to an identifier causing a debugging section to be executed.

The third form of the USE FOR DEBUGGING statement is for referencing file-names:

USE FOR DEBUGGING ON file-name-1 . . .

This debugging section would be executed after any explicit reference to the named file, (e.g., OPEN, CLOSE, READ, DELETE, START). There is a mild inconsistency here due to the syntax and structure of the COBOL language. This statement will not cause the monitoring of all of the activity of the file due to the WRITE and REWRITE statements. With the exception of the WRITE and REWRITE statements, all other statements that are used in file processing explicitly reference the file-name (i.e., READ file-name, OPEN I-O file-name, etc.). The WRITE and REWRITE statements reference the record-name for the file which is an identifier. Therefore, in order to trace all of the activity of a file, each of the records associated with that file must also be referenced in a USE for debugging. This could be done by using one or more USE statements.

There is a fourth option of the USE FOR DEBUGGING statement which is for use in debugging programs utilizing the COBOL teleprocessing capability; it will not be discussed in this paper.

DEBUG-ITEM

The previous paragraphs described the method of establishing a debugging section for procedure-names, identifiers, and file-names. When a debugging section is executed (i.e., the item being monitored is appropriately affected) the debugging system must provide the information necessary for the programmer to debug the program. This is done through a special register generated by the compiler when the compile time debugging switch is 'on'. This special register is accessible in the debugging section by the name of DEBUG-ITEM.

There are six fields subordinate to DEBUG-ITEM which provide the debugging information:

1. DEBUG-LINE—Contains the line number of the source line which caused the debugging section to be executed.
2. DEBUG-NAME—Contains the first thirty characters of the name that caused the debugging section to be executed, e.g., procedure-name-1, identifier-1, or file-name-1. This would include qualifiers and/or subscripts if present but truncated at thirty characters.
3. DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3—There are three occurrences of this item which, and in the case of a data-name which was subscripted/indexed, will contain the occurrence number of each level of the referenced table.
4. DEBUG-CONTENTS—Contains information relative to the name that caused the debugging section to be executed.

- Identifiers—contains the contents of the identifier.
- File-name—contains spaces except that after a READ statement it contains the contents of the record read.
- Procedure-names—contains a variety of information indicating for the most part how control was passed to the procedure-name contained in DEBUG-ITEM:
  START PROGRAM
  (first paragraph).
  SORT INPUT
  SORT OUTPUT
  MERGE OUTPUT
  PERFORM LOOP
  FALL THROUGH
  GO TO
  USE PROCEDURE
  (other than debugging).

The COBOL description of DEBUG-ITEM could be presented as follows:

```
01 DEBUG-ITEM.
   02 DEBUG-LINE PIC X(6).
   02 FILLER PIC X VALUE SPACE.
   02 DEBUG-NAME PIC X(30).
   02 FILLER PIC X VALUE SPACE.
   02 DEBUG-SUB-1 PIC S9(4) SIGN IS LEADING
        SEPARATE CHARACTER.
   02 FILLER PIC X VALUE SPACE.
   02 DEBUG-SUB-2 PIC S9(4) SIGN IS LEADING
        SEPARATE CHARACTER.
   02 FILLER PIC X VALUE SPACE.
   02 DEBUG-SUB-3 PIC S9(4) SIGN IS LEADING
        SEPARATE CHARACTER.
   02 FILLER PIC X VALUE SPACE.
   02 DEBUG-CONTENTS PIC X(n).
```

The X(n) described in the Picture for DEBUG-CONTENTS is the same length as the identifier being debugged or in the case of the procedure-name the length necessary to hold the content of the information placed there by the debugging system.

The following example demonstrates the use of debugging sections for procedure-names and identifiers as well as the results that could be expected.

EXAMPLE:

```
089000 PROCEDURE DIVISION
090000 DECLARATIVES.
090100 SECT1 SECTION.
090200       USE FOR DEBUGGING ON
                PARAGRAPH-X.
090300 DEB-1.
090400       DISPLAY DEBUG-ITEM.
090500 SECT2 SECTION.
090600       USE FOR DEBUGGING ON COUNT-B.
090700 DEB-2.
090800       DISPLAY DEBUG-ITEM.
090900 END DECLARATIVES.
091000 FIRSTSECT SECTION.
```

```
091100 PARAGRAPH-1.
091200         OPEN OUTPUT PRINT-FILE.
          ⋮
101100         MOVE 000000 TO COUNT-B.
101200 PARAGRAPH-X.
101300         ADD 000001 TO COUNT-B.
          ⋮
102100         IF COUNT-B LESS THAN 2 GO TO
                 PARAGRAPH-X.
102200         CLOSE PRINT-FILE.
102300         STOP RUN.
```

The results of the above execution of coding would be:

```
DEBUG  DEBUG-NAME        DEBUG-CONTENTS
LINE
101100 COUNT-B           000000
101200 PARAGRAPH-X       FALL THROUGH
101300 COUNT-B           000001
102100 PARAGRAPH-X       GO TO
101300 COUNT-B           000002
```

In the previous example, had the USE FOR DEBUGGING ON PARAGRAPH-X statement contained the ALL PROCEDURES phrase and the USE FOR DEBUGGING ON COUNT-B statement contained the ALL REFERENCES phrase then the appropriate debugging section would have been executed each time COUNT-B was referenced and for each procedure-name referenced in the program. The following would have been produced:

```
DEBUG  DEBUG-NAME        DEBUG-CONTENTS
LINE
091100 PARAGRAPH-1       START PROGRAM
101100 COUNT-B           000000
101200 PARAGRAPH-X       FALL THROUGH
101300 COUNT-B           000001
102100 COUNT-B           000001
102100 PARAGRAPH-X       GO TO
101300 COUNT-B           000002
102100 COUNT-B           000002
```

CONCLUSION

We conclude this discussion by offering several thoughts with regard to improved software reliability and increased programmer productivity.

1. The advent of new language elements which are designed for debugging in COBOL will permit consideration for the debugging of source programs to be included in the design and programming of an application—not as an afterthought. For knowledge of potential problem areas would permit the inclusion of both debugging sections and debug lines which could provide ample information for producing an adequately "debugged" program.
2. The use of source program preprocessors and assorted post processors including "core dumps" should be for the most part eliminated from the list of tools necessary for the debugging of COBOL programs now that a high level of control can be accomplished through the use of COBOL source statement elements.

REFERENCES

1. Boehm, Barry W., *Some Information Processing Implications of Air Force Space Missions: 1970-1980*, The Rand Corporation, RM-6213-PR, January 1970.
2. *X3.23-1968 American National Standard COBOL*, American National Standards Institute, Inc., 10 East 40th Street, New York, New York, 10016.
3. *X3.23-1974 American National Standard Programming Language COBOL*, American National Standards Institute, Inc., 10 East 40th Street, New York, New York, 10016.

Better manpower utilization using automatic restructuring

by GUY de BALBINE
Caine, Farber and Gordon, Inc.
Pasadena, California

INTRODUCTION

Our intent is to introduce the concept of automatic restructuring as a powerful method for improving the quality of software developed before the advent of structured programming. The quality improvements we are concerned with are neither execution time efficiency nor core size requirements but, rather, higher readability and clear structured code. These, in turn, should improve the reliability and reduce the maintenance costs by making human verification more efficient.

The fact that arbitrary flow diagrams can be mapped into equivalent structured flow diagrams by introducing new Boolean variables has been established by Böhm and Jacopini[2] (see Reference 1 for an example of a program that cannot be restructured without additional Boolean variables). The first steps toward systematizing this mapping are taken in Reference 9.

In practice, however, we have found that adding Boolean variables (whose names are meaningless since they would have to be program generated) makes the code often harder to read. Thus Djikstra's comment[4] that "the exercise to translate an arbitrary flow diagram more or less mechanically into a jumpless one is not to be recommended" because "the resulting flow diagram cannot be expected to be more transparent than the original one."

On the other hand, if we allow certain constrained forms of the GO TO statement, many of the difficulties vanish and readability can be enhanced. One form of the constrained GO TO, which we call UNDO is used to exit from nested structures when necessary, the jump always being a forward jump to the end of a DO group. This is similar to the LEAVE statement in BLISS.[10]

Figure 1 shows an example derived from Reference 1. With the UNDO construct, a natural straightforward representation can be obtained.

Based on the hypothesis that the restructuring process could be applied systematically to existing unstructured programs and enhance their clarity, we have designed and implemented a software tool known as the "structuring engine." We shall now describe in more detail some of our motivations and the experimental results that we have obtained while using the "structuring engine."

IMPROVING THE HUMAN↔SOFTWARE INTERFACE

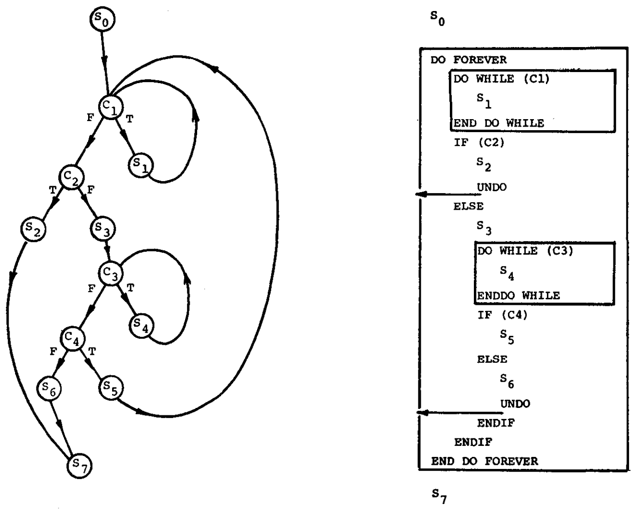
Most production software in existence today was developed using no precise design methodology. The programming languages generally used (FORTRAN, COBOL) were invented over a decade ago and have hardly evolved due to the severe binds imposed by upward compatibility. Maintaining and extending the huge software inventory is a difficult and inefficient task which is becoming even more so year by year. The software documentation is poor, the logic is often obscure, and the authors are most likely to be gone or assigned to other projects. Operational programs still break down with bugs that have managed to escape the most careful scrutiny. Modifications and extensions are dreaded and postponed since they are likely to cause perturbations whose far ranging effects cannot be easily and reliably assessed.

We do not claim to have a panacea that can cure all of these problems instantly. However, the experience gained while developing large scale software using structured programming has shown some of the important factors that influence software reliability and maintenance costs. In our experience, the quality of the human↔software interface is one such factor since it influences the efficiency of all manhours invested at the program level, both during development and maintenance.

To benefit from a better human ↔ software interface applicable to future software development, as well as to current software, we suggest extending commonly available programming languages, imposing some constraints to ensure proper language usage, emphasizing the need for visual improvement of programs, and providing transitional tools to assist in the conversion of existing software to meet the new interface specifications.

Language extensions

The only precise, and by definition up-to-date, source of internal documentation for most software in existence today lies in the programs themselves. Understanding what programs accomplish implies an understanding in



```
s0
DO FOREVER
    DO WHILE (C1)
        S1
    END DO WHILE
    IF (C2)
        S2
        UNDO
    ELSE
        S3
        DO WHILE (C3)
            S4
        ENDDO WHILE
        IF (C4)
            S5
        ELSE
            S6
            UNDO
        ENDIF
    ENDIF
END DO FOREVER
s7
```

Figure 1—Example of UNDO usage

the formalism and at the level of detail imposed by the programming language used as a vehicle for implementation. Thus, any shortcomings of the implementation language have a direct impact on the effort needed to understand what the programs do and to modify and extend them successfully.

The two most widespread programming languages, FORTRAN and COBOL, do not contain adequate mechanisms to support structured coding. The limited facilities they provide can be exploited very cleverly to look somewhat like structured code. However, a substantial effort is needed to maintain proper indentation and the legibility is never as good as that obtainable with a structured language.

The obvious step is to build preprocessors to provide the necessary syntactic extensions and perform some of the manual chores such as automatic indentation. Several dozen preprocessors have already been built to translate various brands of structured FORTRAN into pure FORTRAN.[7] Our effort along these lines has led to the design and implementation of the S-FORTRAN language and translator. S-FORTRAN embodies a small but powerful set of structured constructs. S-FORTRAN was designed to serve both as a target language for restructured programs and as an implementation language for new programs. It is not only simple but easy to remember unambiguously. The S-FORTRAN language is succinctly described in the Appendix.

We do not wish at this point to discuss at length the individual merits of each S-FORTRAN feature and whether LOOP is a better term than DO FOREVER or should DO UNTIL test first rather than execute first. These decisions are mostly conventions. Let us simply express the hope that a consensus will soon develop so that a "de facto" standard will prevail. Structured FORTRAN programs will then be unambiguously understood by all.

Language usage

Providing extended languages to permit structured coding is not sufficient to guarantee software clarity. Programmers can still misuse structured languages to follow their traditional thought processes, the result being obscure programs under the guise of structured code.

Rather than resort to building enforcement tools, it is our belief that the simplicity and intellectual appeal of a well formed program will generate the necessary motivation among programmers to adopt a new standard of quality.

Visual improvement

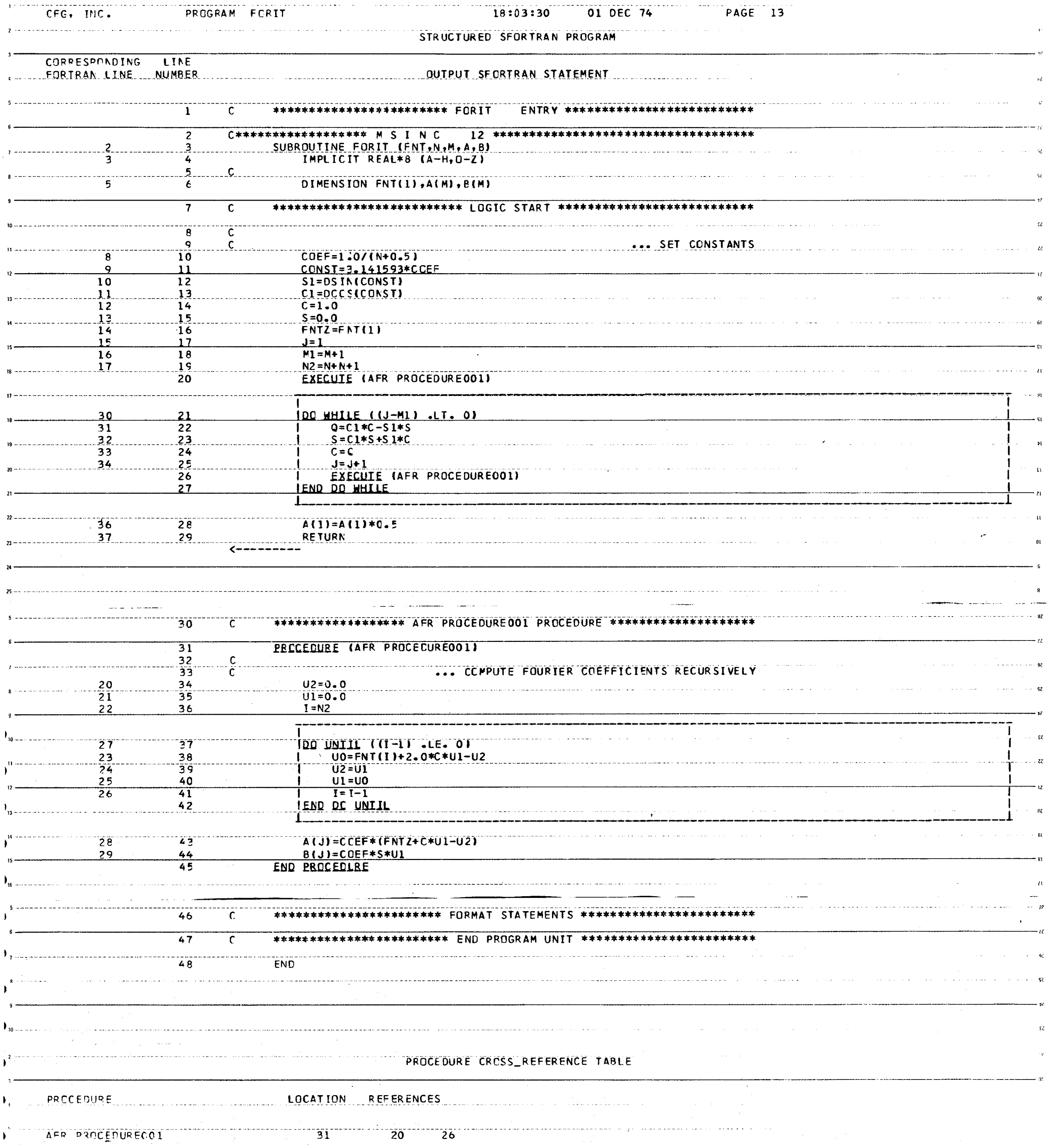
Structured coding techniques require that programs be systematically indented to stress the relationships between code segments. This hierarchical arrangement allows a quick grasp of the global as well as the local structure of the code. Understanding the code no longer requires keeping track of many scattered items such as labels and transfers. Rather, it means perceiving visual patterns that can be precisely mapped into our analytical understanding of the solution. Each part and subpart corresponds to a block of code, carefully delimited to facilitate its verification. Systematic indentation makes it easy to collect the conditions controlling the execution of each indentation level down to the code segment being examined.

The power of visual perception can be readily tapped by developing patterns whenever feasible. Symmetry, lack of symmetry, block indentation, regularity, recurring patterns, aligned similar items, . . . are characteristics that can be detected at a glance by the eye. Interestingly enough, these are characteristics whose global nature is usually hard to detect and utilize automatically with software tools.

HOW TO BENEFIT FROM THE NEW INTERFACE

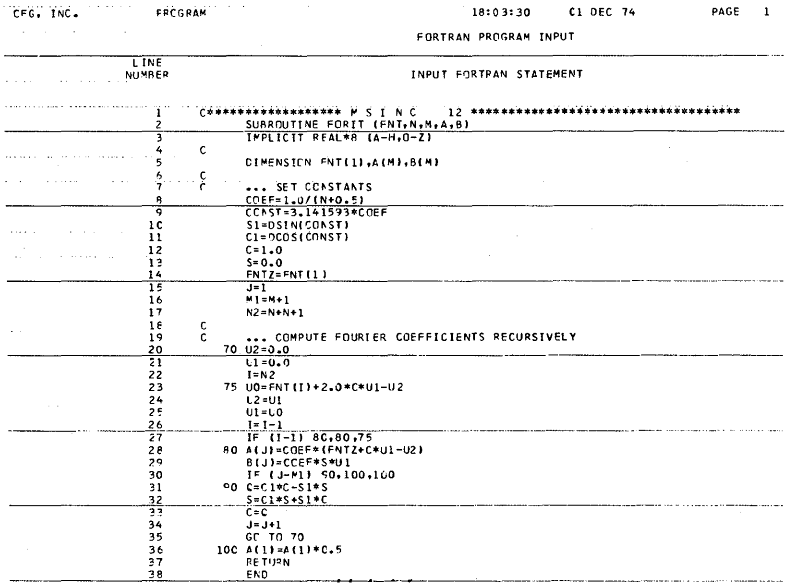
Formulating a better human↔software interface is clearly valuable for software that has not yet been written. The important point is that such an improvement can also be applied to a large part of the software inventory in existence today. It is our belief that this should lead to a significant reduction in the maintenance effort by better utilizing the available manpower.

Until very recently, the main route for modernizing existing unstructured software was to start over with a clean top-down design and structured implementation. Needless to say, such complete manual reprogramming should not be undertaken without a very careful evaluation of the potential gain versus the effort involved. We have found that the major obstacles to manual reprogramming are the need for top talent during the redesign phase, the manpower expenditure, the elapsed time before a new



```
CFG, INC.            PROGRAM  FCRIT                                   18:03:30      01 DEC 74          PAGE  13
                                                           STRUCTURED SFORTRAN PROGRAM

CORRESPONDING   LINE
FORTRAN LINE   NUMBER                                      OUTPUT SFORTRAN STATEMENT

                   1      C      ************************ FORIT    ENTRY *************************

                   2      C***************** M S I N C    12 ************************************
     2             3             SUBROUTINE FORIT (FNT,N,M,A,B)
     3             4                  IMPLICIT REAL*8 (A-H,O-Z)
                   5      C
     5             6                  DIMENSION FNT(1),A(M),B(M)

                   7      C      ************************** LOGIC START **************************

                   8      C
                   9      C                                                      ... SET CONSTANTS
     8            10                  COEF=1.0/(N+0.5)
     9            11                  CONST=3.141593*CCEF
    10            12                  S1=DSIN(CONST)
    11            13                  C1=DCCS(CONST)
    12            14                  C=1.0
    13            15                  S=0.0
    14            16                  FNTZ=FNT(1)
    15            17                  J=1
    16            18                  M1=M+1
    17            19                  N2=N+N+1
                  20                  EXECUTE (AFR PROCEDURE001)
                                     ----------------------------------------------------------
                                     I                                                        I
    30            21                 IDO WHILE ((J-M1) .LT. 0)                                I
    31            22                 I    Q=C1*C-S1*S                                         I
    32            23                 I    S=C1*S+S1*C                                         I
    33            24                 I    C=C                                                 I
    34            25                 I    J=J+1                                               I
                  26                 I    EXECUTE (AFR PROCEDURE001)                          I
                  27                 IEND DO WHILE                                            I
                                     I--------------------------------------------------------I

    36            28                  A(1)=A(1)*0.5
    37            29                  RETURN
                          <---------



                  30      C      ***************** AFR PROCEDURE001 PROCEDURE ********************

                  31             PRCCEDURE (AFR PROCECURE001)
                  32      C
                  33      C                                 ... CCMPUTE FOURIER COEFFICIENTS RECURSIVELY
    20            34                  U2=0.0
    21            35                  U1=0.0
    22            36                  I=N2
                                     ----------------------------------------------------------
                                     I                                                        I
    27            37                 IDO UNTIL ((I-1) .LE. 0)                                 I
    23            38                 I    U0=FNT(I)+2.0*C*U1-U2                               I
    24            39                 I    U2=U1                                               I
    25            40                 I    U1=U0                                               I
    26            41                 I    I=I-1                                               I
                  42                 IEND DC UNTIL                                            I
                                     I--------------------------------------------------------I

    28            43                  A(J)=CCEF*(FNTZ+C*U1-U2)
    29            44                  B(J)=COEF*S*U1
                  45             END PROCEDLRE


                  46      C      *********************** FORMAT STATEMENTS ***********************

                  47      C      *********************** END PROGRAM UNIT ************************

                  48             END



                                                           PROCEDURE CRCSS_REFERENCE TABLE

PRCCEDURE                                 LOCATION   REFERENCES

AFR PROCEDURE001                             31          20       26
```

Figure 2—Sample program FORIT before and after restructuring



```
CFG, INC.          FRCGRAM                                 18:03:30     C1 DEC 74          PAGE   1
                                                           FORTRAN PROGRAM INPUT

               LINE
               NUMBER                                      INPUT FORTRAN STATEMENT

                 1     C***************** M S I N C    12 ************************************
                 2             SUBROUTINE FORIT (FNT,N,M,A,B)
                 3             IMPLICIT REAL*8 (A-H,O-Z)
                 4     C
                 5             DIMENSICN FNT(1),A(M),B(M)
                 6     C
                 7     C       ... SET CCNSTANTS
                 8             COEF=1.0/(N+0.5)
                 9             CCNST=3.141593*COEF
                1C             S1=DSIN(CONST)
                11             C1=DCOS(CONST)
                12             C=1.0
                13             S=0.0
                14             FNTZ=FNT(1)
                15             J=1
                16             M1=M+1
                17             N2=N+N+1
                1E     C
                19     C       ... COMPUTE FOURIER COEFFICIENTS RECURSIVELY
                20          70 U2=0.0
                21             U1=0.0
                22             I=N2
                23          75 U0=FNT(I)+2.0*C*U1-U2
                24             U2=U1
                25             U1=U0
                26             I=I-1
                27             IF (I-1) 8C,80,75
                28          80 A(J)=COEF*(FNTZ+C*U1-U2)
                29             B(J)=CCEF*S*U1
                30             IF (J-M1) 90,100,100
                31          90 C=C1*C-S1*S
                32             S=C1*S+S1*C
                33             C=C
                34             J=J+1
                35             GC TO 70
                36         10C A(1)=A(1)*0.5
                37             RETURN
                38             END
```

system is up and running and the penalty for having to go through a full testing and debugging period again.

As an alternative, we have developed a method which is much easier to apply in practice. It consists in keeping the global design as is, in particular the data structures, and in automatically transforming every program into an equivalent structured program, visually improved to make its reading easier and its understanding more thorough. This method is supported by a software tool known as the "structuring engine." We have applied this tool to a variety of FORTRAN programs. We shall now describe the characteristics of the tool and some of the experimental results that we have obtained so far.

THE "STRUCTURING ENGINE"

Capabilities

The "structuring engine," as it now exists, is a large task running on an IBM/370 under VS. It consists of over 30,000 lines of structured PL/1 code. It will restructure programs written in FORTRAN including any language extensions acceptable by IBM, Univac, CDC and Honeywell compilers.

Each program or subprogram is restructured independently. The complete flow graph of each program or subprogram is analyzed to determine the best strategy for obtaining a well structured program. Machine dependencies are taken into account when building the flow graph because the interpretation of some statements depends on the particular compiler that the program was intended for. For instance, values outside the range of a computed GO TO can be handled in three distinct ways depending upon the particular compiler implementation. Such variations are taken into account by the "structuring engine" which generates the necessary statements to guarantee consistency in the restructured output.

In general, the restructured programs will bear little resemblance to the original unstructured ones, particularly if the logic was complex and somewhat twisted to start with. In the output, the logic flows from top to bottom, from the single entry to the single exit.

Figure 2 is an example of a simple program before and after restructuring. Similarly, Figure 3 shows what happens in the case of a heavily folded program.

The restructured programs are equivalent to those from which they are derived in the sense that they behave identically at run time. That is, they carry out the same sequence of operations on the data structures, great care being taken that the ordering of operations not be modified. For instance, a three way arithmetic IF cannot be simply converted into two nested S-FORTRAN IF statements because the arithmetic expression would then be evaluated twice. In that case, incorrect results might be obtained if the arithmetic expression contains calls to abnormal functions i.e., functions which do not always produce the same results from a given set of inputs.

One of the basic processes used in restructuring is known as node splitting. If a node of the subgraph can be reached from two different paths that must be separated, the node is split into two identical nodes so that each path can have its own copy of the node.

If the node splitting operations were carried out indiscriminately, the resulting S-FORTRAN programs would often become so large as to be virtually useless. Not only would clarity be lost but the object program would be likely not to fit in the target machine. To circumvent that difficulty, the "structuring engine" tries recognizing proper subgraphs that can be turned into procedures instead of being duplicated in line. A procedure is simply a section of code with one entry and one exit. This concept corresponds to the PERFORMed group in COBOL, but with additional constraints to guarantee a clean invocation and a clean return. Once a procedure has been extracted and given a name, it can be referenced from many locations within the restructured program, including from other procedures. The example in Figure 2 contains one procedure, the one in Figure 3 contains two. The decision whether to expand code in line or create procedures can be externally controlled using a threshold which indicates how complex a subgraph must be before it becomes a procedure. Procedures are not separate subprograms. Rather, they are segments of code that can be executed from various locations within a particular program or subprogram. The EXECUTE command hides the ASSIGNed GO TO linkage that a FORTRAN programmer would have to set up otherwise.

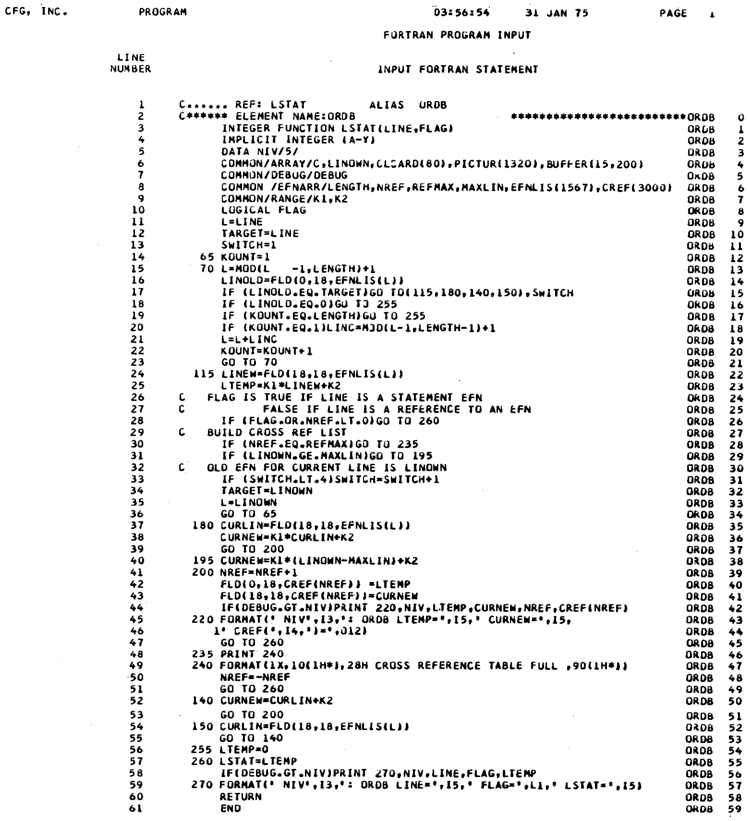
To visually improve the resulting code, every statement is laid out according to its logical indentation level. This stresses its relationship with other statements in the same program unit. A box is built around each complete DO group to enhance the scope of the DO statements. Statements such as UNDO, CYCLE, and RETURN are followed by an arrow that attracts the attention of the reader and shows him immediately what implications these statements have on the logic flow. Consecutive comment cards are right adjusted by block in order to make them as unobtrusive as possible.

Of course, if the modules to be restructured contain logic errors, the same errors will be found in the structured output. In general, the "structuring engine" is incapable of detecting errors except for some obvious language violations. The input programs are supposed to have compiled correctly so that the errors we are really trying to eliminate are errors in the logic that cannot be identified without an intimate understanding of the problem. Only a programmer aware of the problem being solved can discover and correct these errors.

Emphasis in building the "structuring engine" has been on reliability rather than efficiency. This has been achieved through a combination of structured design techniques, self identifiable data structures and dynamic assertion verification at run time i.e., the constant verification that the assumptions underlying the design are never violated during production runs.

Experimental results

We are currently applying the "structuring engine" to a wide variety of unstructured FORTRAN code. Although our analysis is far from complete, we would like to comment on some of the experimental results that we have obtained so far.



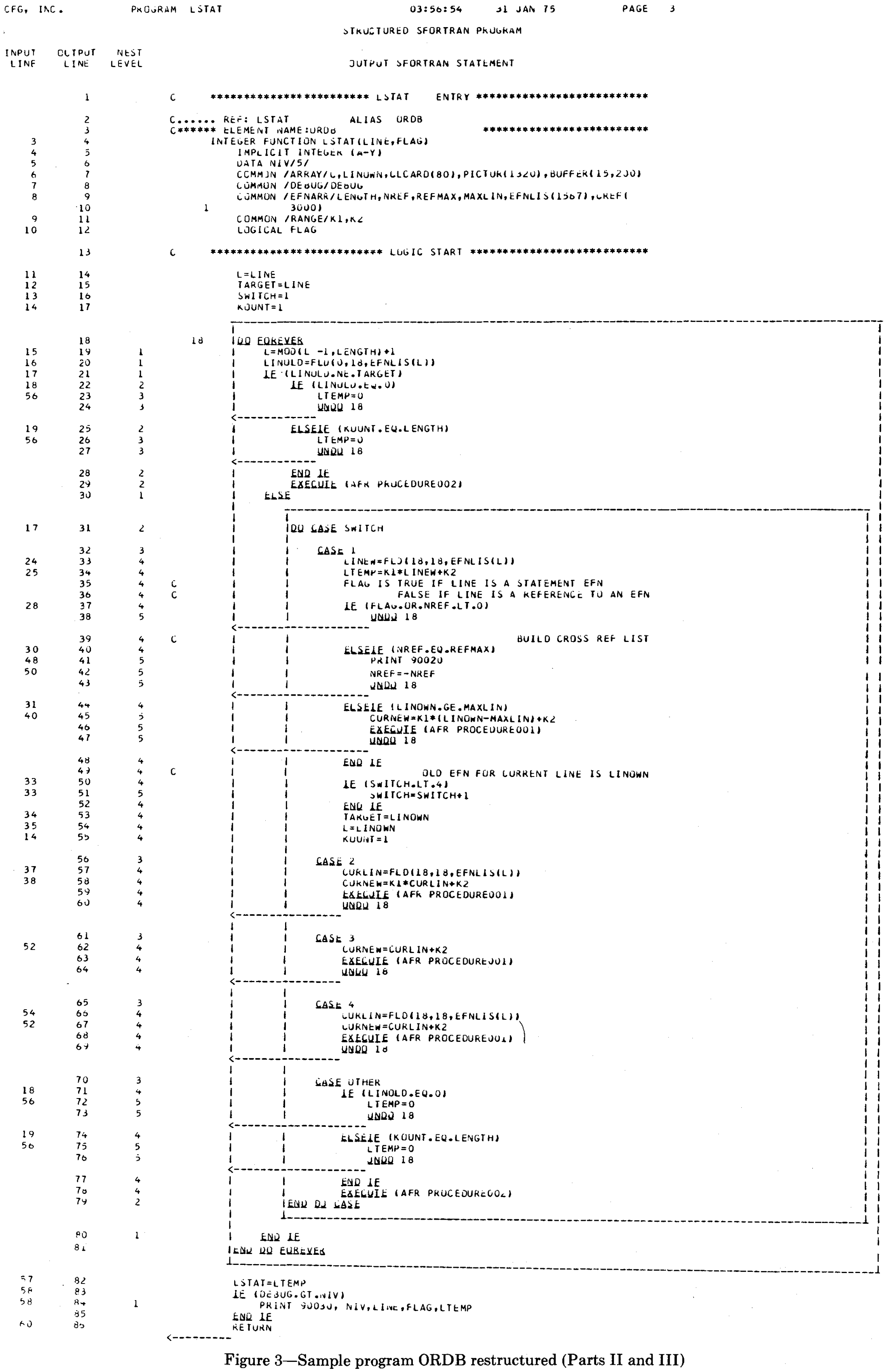
```
CFG, INC.        PROGRAM                          03:56:54    31 JAN 75        PAGE  1
                                          FORTRAN PROGRAM INPUT

          LINE
          NUMBER                          INPUT FORTRAN STATEMENT

     C...... REF: LSTAT        ALIAS  URDB
     C****** ELEMENT NAME:ORDB                          *************************ORDB  0
           INTEGER FUNCTION LSTAT(LINE,FLAG)                                      ORDB  1
           IMPLICIT INTEGER (A-Y)                                                 ORDB  2
           DATA NIV/5/                                                            ORDB  3
           COMMON/ARRAY/C,LINOWN,CLCARD(80),PICTUR(1320),BUFFER(15,200)           ORDB  4
           COMMON/DEBUG/DEBUG                                                     ORDB  5
           COMMON /EFNARR/LENGTH,NREF,REFMAX,MAXLIN,EFNLIS(1567),CREF(3000)       ORDB  6
           COMMON/RANGE/K1,K2                                                     ORDB  7
           LOGICAL FLAG                                                           ORDB  8
           L=LINE                                                                 ORDB  9
           TARGET=LINE                                                            ORDB 10
           SWITCH=1                                                               ORDB 11
        65 KOUNT=1                                                                ORDB 12
        70 L=MOD(L   -1,LENGTH)+1                                                 ORDB 13
           LINOLD=FLD(0,18,EFNLIS(L))                                             ORDB 14
           IF (LINOLD.EQ.TARGET)GO TO(115,180,140,150),SWITCH                     ORDB 15
           IF (LINOLD.EQ.0)GU TJ 255                                              ORDB 16
           IF (KOUNT.EQ.LENGTH)GU TO 255                                          ORDB 17
           IF (KOUNT.EQ.1)LINC=MJD(L-1,LENGTH-1)+1                                ORDB 18
           L=L+LINC                                                               ORDB 19
           KOUNT=KOUNT+1                                                          ORDB 20
           GO TO 70                                                               ORDB 21
       115 LINEW=FLD(18,18,EFNLIS(L))                                             ORDB 22
           LTEMP=K1*LINEW+K2                                                      ORDB 23
     C   FLAG IS TRUE IF LINE IS A STATEMENT EFN                                  ORDB 24
     C            FALSE IF LINE IS A REFERENCE TO AN EFN                          ORDB 25
           IF (FLAG.OR.NREF.LT.0)GO TO 260                                        ORDB 26
     C   BUILD CROSS REF LIST                                                     ORDB 27
           IF (NREF.EQ.REFMAX)GO TO 235                                           ORDB 28
           IF (LINOWN.GE.MAXLIN)GO TO 195                                         ORDB 29
     C   OLD EFN FOR CURRENT LINE IS LINOWN                                       ORDB 30
           IF (SWITCH.LT.4)SWITCH=SWITCH+1                                        ORDB 31
           TARGET=LINOWN                                                          ORDB 32
           L=LINOWN                                                               ORDB 33
           GO TO 65                                                               ORDB 34
       180 CURLIN=FLD(18,18,EFNLIS(L))                                            ORDB 35
           CURNEW=K1*CURLIN+K2                                                    ORDB 36
           GO TO 200                                                              ORDB 37
       195 CURNEW=K1*(LINOWN-MAXLIN)+K2                                           ORDB 38
       200 NREF=NREF+1                                                            ORDB 39
           FLD(0,18,CREF(NREF)) =LTEMP                                            ORDB 40
           FLD(18,18,CREF(NREF))=CURNEW                                           ORDB 41
           IF(DEBUG.GT.NIV)PRINT 220,NIV,LTEMP,CURNEW,NREF,CREF(NREF)             ORDB 42
       220 FORMAT(' NIV',I3,': ORDB LTEMP=',I5,' CURNEW=',I5,                     ORDB 43
          1' CREF(',I4,')=',O12)                                                  ORDB 44
           GO TO 260                                                              ORDB 45
       235 PRINT 240                                                              ORDB 46
       240 FORMAT(1X,10(1H*),28H CROSS REFERENCE TABLE FULL ,90(1H*))             ORDB 47
           NREF=-NREF                                                             ORDB 48
           GO TO 260                                                              ORDB 49
       140 CURNEW=CURLIN+K2                                                       ORDB 50
           GO TO 200                                                              ORDB 51
       150 CURLIN=FLD(18,18,EFNLIS(L))                                            ORDB 52
           GO TO 140                                                              ORDB 53
       255 LTEMP=0                                                                ORDB 54
       260 LSTAT=LTEMP                                                            ORDB 55
           IF(DEBUG.GT.NIV)PRINT 270,NIV,LINE,FLAG,LTEMP                          ORDB 56
       270 FORMAT(' NIV',I3,': ORDB LINE=',I5,' FLAG=',L1,' LSTAT=',I5)           ORDB 57
           RETURN                                                                 ORDB 58
           END                                                                    ORDB 59
```

Figure 3—Sample program ORDB unstructured (Part I)

Clarity of the restructured programs

A reliable assessment of clarity improvements is obviously quite difficult to obtain until we get some figures on maintenance costs. The familiarity of the end user with structured code is a factor as noted in Reference 6. The cleverness of the "structuring engine" in making the right choices is obviously another important factor since there are not one but many solutions to the restructuring problem. So far, we have found that:

- the majority of the programs (about 90 percent) will come out extremely clear, at least in our opinion and in that of end users that have worked with restructured programs.
- the rest (about 10 percent) will either remain complex or become lengthy or both. In this group, we find a number of programs that could be handled more cleverly by the "structuring engine" and, therefore, move into the above category. We are obviously building the necessary improvements into the "structuring engine." Still, there are some programs that will probably never look very good. They are ill-designed. The problem that they are supposed to solve should be reexamined and a complete redesign and reprogramming of these programs may be necessary.



```
CFG, INC.           PROGRAM  LSTAT                             03:56:54      31 JAN 75          PAGE    3

                                                     STRUCTURED SFORTRAN PROGRAM

INPUT   OUTPUT    NEST
 LINE     LINE   LEVEL                                OUTPUT SFORTRAN STATEMENT

             1            C     ************************ LSTAT    ENTRY *************************

             2            C...... REF: LSTAT            ALIAS  ORDB
             3            C****** ELEMENT NAME:ORDB                      *************************
     3       4                  INTEGER FUNCTION LSTAT(LINE,FLAG)
     4       5                      IMPLICIT INTEGER (A-Y)
     5       6                      DATA NIV/5/
     6       7                      COMMON /ARRAY/C,LINOWN,CLCARD(80),PICTUR(1320),BUFFER(15,200)
     7       8                      COMMON /DEBUG/DEBUG
     8       9                      COMMON /EFNARR/LENGTH,NREF,REFMAX,MAXLIN,EFNLIS(1567),CREF(
            10                 1         3000)
     9      11                      COMMON /RANGE/K1,K2
    10      12                      LOGICAL FLAG

            13            C     ************************* LOGIC START **************************

    11      14                      L=LINE
    12      15                      TARGET=LINE
    13      16                      SWITCH=1
    14      17                      KOUNT=1
                                    ------------------------------------------------------------------------------------------------------
                                    |                                                                                                    |
            18               18     |DO FOREVER                                                                                          |
    15      19       1              |    L=MOD(L -1,LENGTH)+1                                                                            |
    16      20       1              |    LINOLD=FLD(0,18,EFNLIS(L))                                                                      |
    17      21       1              |    IF (LINOLD.NE.TARGET)                                                                           |
    18      22       2              |        IF (LINOLD.EQ.0)                                                                            |
    56      23       3              |            LTEMP=0                                                                                 |
            24       3              |            UNDO 18                                                                                 |
                                    <-----------                                                                                         |
    19      25       2              |        ELSEIF (KOUNT.EQ.LENGTH)                                                                    |
    56      26       3              |            LTEMP=0                                                                                 |
            27       3              |            UNDO 18                                                                                 |
                                    <-----------                                                                                         |
            28       2              |        END IF                                                                                      |
            29       2              |        EXECUTE (AFR PROCEDURE002)                                                                  |
            30       1              |    ELSE                                                                                            |
                                    |                                                                                                    |
                                    |       -------------------------------------------------------------------------------------------- |
                                    |       |                                                                                          | |
    17      31       2              |       |DO CASE SWITCH                                                                            | |
                                    |       |                                                                                          | |
            32       3              |       |    CASE 1                                                                                | |
    24      33       4              |       |        LINEW=FLD(18,18,EFNLIS(L))                                                        | |
    25      34       4              |       |        LTEMP=K1*LINEW+K2                                                                 | |
            35       4    C         |       |        FLAG IS TRUE IF LINE IS A STATEMENT EFN                                           | |
            36       4    C         |       |                FALSE IF LINE IS A REFERENCE TO AN EFN                                    | |
    28      37       4              |       |        IF (FLAG.OR.NREF.LT.0)                                                            | |
            38       5              |       |            UNDO 18                                                               | |
                                    <--------------------                                                                              | |
            39       4    C         |       |                                   BUILD CROSS REF LIST                                   | |
    30      40       4              |       |        ELSEIF (NREF.EQ.REFMAX)                                                           | |
    48      41       5              |       |            PRINT 90020                                                                   | |
    50      42       5              |       |            NREF=-NREF                                                                    | |
            43       5              |       |            UNDO 18                                                                       | |
                                    <--------------------                                                                              | |
    31      44       4              |       |        ELSEIF (LINOWN.GE.MAXLIN)                                                         | |
    40      45       5              |       |            CURNEW=K1*(LINOWN-MAXLIN)+K2                                                  | |
            46       5              |       |            EXECUTE (AFR PROCEDURE001)                                                    | |
            47       5              |       |            UNDO 18                                                                       | |
                                    <--------------------                                                                              | |
            48       4              |       |        END IF                                                                            | |
            49       4    C         |       |                    OLD EFN FOR CURRENT LINE IS LINOWN                                    | |
    33      50       4              |       |        IF (SWITCH.LT.4)                                                                  | |
    33      51       5              |       |            SWITCH=SWITCH+1                                                               | |
            52       4              |       |        END IF                                                                            | |
    34      53       4              |       |        TARGET=LINOWN                                                                     | |
    35      54       4              |       |        L=LINOWN                                                                          | |
    14      55       4              |       |        KOUNT=1                                                                           | |
                                    |       |                                                                                          | |
            56       3              |       |    CASE 2                                                                                | |
    37      57       4              |       |        CURLIN=FLD(18,18,EFNLIS(L))                                                       | |
    38      58       4              |       |        CURNEW=K1*CURLIN+K2                                                               | |
            59       4              |       |        EXECUTE (AFR PROCEDURE001)                                                        | |
            60       4              |       |        UNDO 18                                                                           | |
                                    <----------------                                                                                  | |
                                    |       |                                                                                          | |
            61       3              |       |    CASE 3                                                                                | |
    52      62       4              |       |        CURNEW=CURLIN+K2                                                                  | |
            63       4              |       |        EXECUTE (AFR PROCEDURE001)                                                        | |
            64       4              |       |        UNDO 18                                                                           | |
                                    <----------------                                                                                  | |
                                    |       |                                                                                          | |
            65       3              |       |    CASE 4                                                                                | |
    54      66       4              |       |        CURLIN=FLD(18,18,EFNLIS(L))                                                       | |
    52      67       4              |       |        CURNEW=CURLIN+K2                                                                  | |
            68       4              |       |        EXECUTE (AFR PROCEDURE001)                                                        | |
            69       4              |       |        UNDO 18                                                                           | |
                                    <----------------                                                                                  | |
                                    |       |                                                                                          | |
            70       3              |       |    CASE OTHER                                                                            | |
    18      71       4              |       |        IF (LINOLD.EQ.0)                                                                  | |
    56      72       5              |       |            LTEMP=0                                                                       | |
            73       5              |       |            UNDO 18                                                                       | |
                                    <--------------------                                                                              | |
    19      74       4              |       |        ELSEIF (KOUNT.EQ.LENGTH)                                                          | |
    56      75       5              |       |            LTEMP=0                                                                       | |
            76       5              |       |            UNDO 18                                                                       | |
                                    <--------------------                                                                              | |
            77       4              |       |        END IF                                                                            | |
            78       4              |       |        EXECUTE (AFR PROCEDURE002)                                                        | |
            79       2              |       |END DO CASE                                                                               | |
                                    |       |------------------------------------------------------------------------------------------| |
                                    |                                                                                                    |
            80       1              |    END IF                                                                                          |
            81                      |END DO FOREVER                                                                                      |
                                    |----------------------------------------------------------------------------------------------------|

    57      82                      LSTAT=LTEMP
    58      83                      IF (DEBUG.GT.NIV)
    58      84       1                  PRINT 90030, NIV,LINE,FLAG,LTEMP
            85                      END IF
    60      86                      RETURN
                         <---------
```

Figure 3—Sample program ORDB restructured (Parts II and III)

Execution characteristics of the restructured programs

Let us now try to answer some of the most common questions regarding this automatic restructuring process. What price do we pay for the improved clarity of the restructured programs? In particular, how do the restructured programs execute compared to the original ones?

To answer these questions, we must first examine the various components in the processing chain as shown in Figure 4. The "structuring engine" transforms unstructured FORTRAN into structured S-FORTRAN. The resulting S-FORTRAN programs are then translated back into FORTRAN using the S-FORTRAN to FORTRAN translator. At that point, we have pure FORTRAN source code again which can be compiled, loaded and executed. Thus, the characteristics that we are reporting on involve not only the "structuring engine" but also the translator and a compiler.

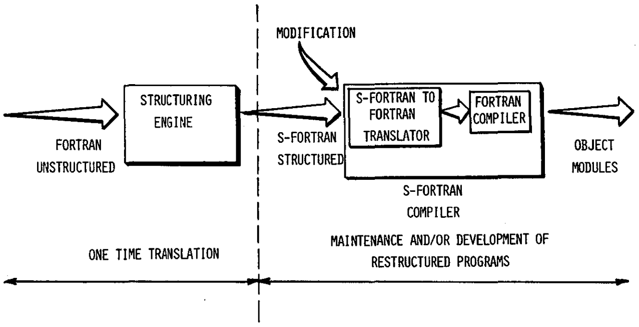


Figure 4—The restructuring chain

The core size of the object modules produced from the restructured programs has been found to always be larger than that of the original modules, typically by about 20 percent.

We know from Reference 9 that arbitrary programs cannot be restructured without increasing their running time or their core size. In the present case, we have chosen to accept a limited increase in memory size. The creation of internal procedures is our method for preventing a program from growing beyond an acceptable point.

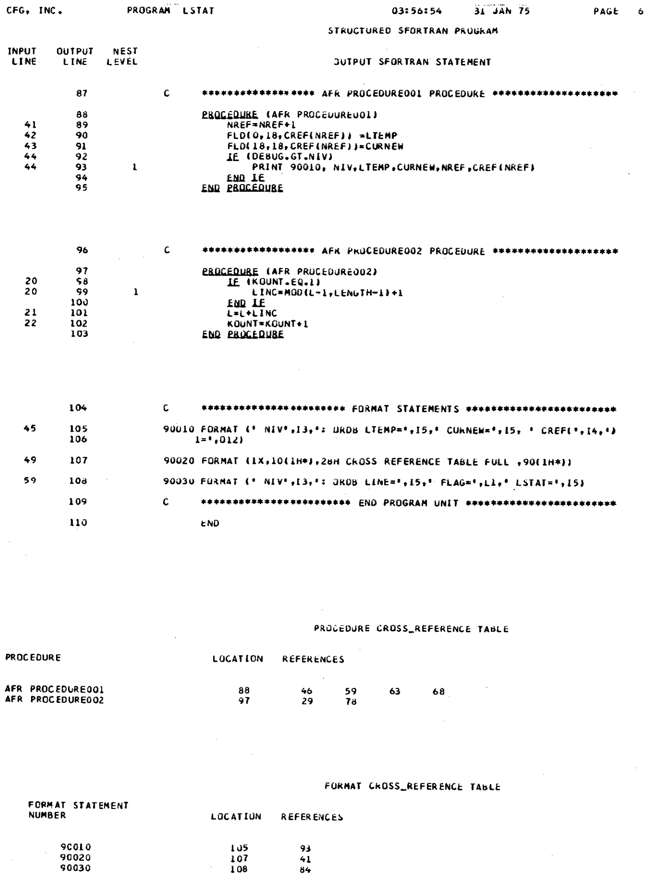

```
CFG, INC.          PROGRAM  LSTAT                            03:56:54     31 JAN 75          PAGE   6
                                                       STRUCTURED SFORTRAN PROGRAM

INPUT   OUTPUT   NEST
 LINE    LINE    LEVEL                                 OUTPUT SFORTRAN STATEMENT

          87              C      ****************** AFR PROCEDURE001 PROCEDURE *******************

          88                     PROCEDURE (AFR PROCEDURE001)
  41      89                          NREF=NREF+1
  42      90                          FLD(0,18,CREF(NREF)) =LTEMP
  43      91                          FLD(18,18,CREF(NREF))=CURNEW
  44      92                          IF (DEBUG.GT.NIV)
  44      93        1                      PRINT 90010, NIV,LTEMP,CURNEW,NREF,CREF(NREF)
          94                          END IF
          95                     END PROCEDURE


          96              C      ****************** AFR PROCEDURE002 PROCEDURE *******************

          97                     PROCEDURE (AFR PROCEDURE002)
  20      98                          IF (KOUNT.EQ.1)
  20      99        1                      LINC=MOD(L-1,LENGTH-1)+1
         100                          END IF
  21     101                          L=L+LINC
  22     102                          KOUNT=KOUNT+1
         103                     END PROCEDURE


         104              C      *********************** FORMAT STATEMENTS ***********************

  45     105              90010 FORMAT (' NIV',I3,': UROB LTEMP=',I5,' CURNEW=',I5, ' CREF(',I4,')
         106                    1=',O12)
  49     107              90020 FORMAT (1X,10(1H*),28H CROSS REFERENCE TABLE FULL ,90(1H*))
  59     108              90030 FORMAT (' NIV',I3,': UROB LINE=',I5,' FLAG=',L1,' LSTAT=',I5)
         109              C      *********************** END PROGRAM UNIT ***********************
         110                     END


                                                     PROCEDURE CROSS_REFERENCE TABLE

PROCEDURE                           LOCATION    REFERENCES

AFR PROCEDURE001                       88          46       59       63       68
AFR PROCEDURE002                       97          29       78


                                                      FORMAT CROSS_REFERENCE TABLE

  FORMAT STATEMENT
  NUMBER                            LOCATION    REFERENCES

       90010                           105         93
       90020                           107         41
       90030                           108         84
```

Figure 3—Continued

Figure 5 shows a typical distribution of core size expansion ratios (1 would mean no increase) as a function of the size of the object module for the unstructured program when compiled with IBM's FORTRAN G compiler. The circled data point corresponds to a program that the "structuring engine" could not structure without producing three times as many S-FORTRAN cards as there were FORTRAN cards. This "abnormal" expansion factor was caused by a deeply nested section of code that could not be turned into procedure because it would then have contained an UNDO outside its scope. Such an UNDO outside the scope of a procedure is not permitted in S-FORTRAN. Consequently, the same section of code was duplicated 18 times throughout the program.

Data on the execution speed of the restructured programs has been harder to get because most of the programs we have restructured so far were components of much larger systems which we could not run ourselves. Preliminary results show that we should expect slight variations in the running time with a trend toward a re-

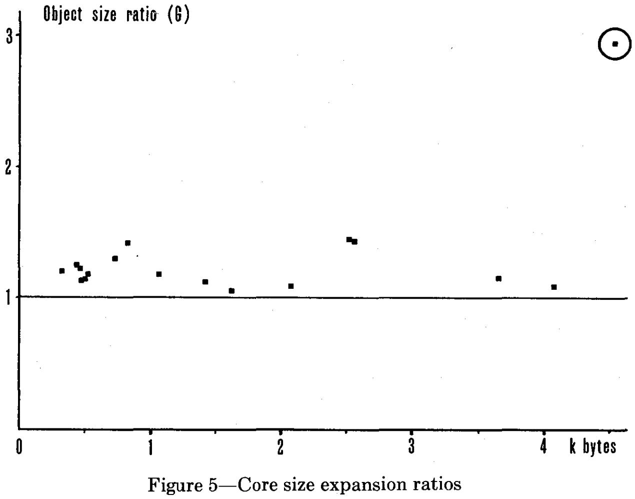


Figure 5—Core size expansion ratios

duction rather than an increase. This may seem paradoxical at first but can be explained as follows. There are two major factors that influence the running time in opposite ways:

(i) the size of the basic block: the restructuring process cannot decrease the average size of the basic block and in general will increase it. Thus, an optimizing compiler should generate better code within each basic block of the restructured programs.
(ii) the control flow statements produced by the translator to support branching, looping and procedure referencing: These require, in general, more instructions than needed to implement the original control logic.

The first factor tends to make restructured programs run faster whereas the latter tends to slow them down. This means that with a translator generating very good code we should be able to have programs run faster restructured than unstructured. In fact, we have now built more sophistication into the translator than had been originally planned in order to make full use of the capabilities of optimizing compilers. For instance, with IBM's FORTRAN H (OPT=2) compiler, changes in the translation of IF statements have reduced in one instance the core size by 12 percent and the running time by 8 percent when compared with earlier versions of the translator.

CONCLUSIONS

Automatic restructuring as implemented by the "structuring engine" is proposed as a method to modernize existing programs. It should prove much more practical than manual reprogramming, particularly with regards to manpower requirements, conversion time and the reliability of the conversion process itself.

Manpower requirements are reduced since no major human effort is invested redoing what already exists. On the contrary, programmer time is devoted to perusing restructured programs, implementing improvements wherever deficiencies show up, and correcting errors. In particular, any program which still appears to be overly complex after restructuring compared to what it is supposed to accomplish, becomes a good candidate for an in-depth investigation of the reasons underlying its apparent complexity. Poor algorithms may be pinned down fast and replaced accordingly. The overall result is that the programmer understands the structured code more rapidly and can, therefore, allocate more time to difficult areas. Consequently, his error detection rate increases, thus justifying our claim to improved software reliability.

Conversion time is negligible compared to that required for manual reprogramming. In particular, the project's clock is not set back since the restructuring process does not introduce any new errors.

Of course, there may still be cases where complete redesign and reprogramming appear to be absolutely necessary. Under those circumstances, the "structuring engine" can still play an important role. Indeed, no matter how unstructured and clumsy the original software may be, it represents an approximate solution to the problem, correct in most if not all of the cases. As such, it acts as a repository for a wealth of details that were added throughout the life cycle of the software to handle unusual and certainly unforeseen cases. Starting from this rich data, the "structuring engine" becomes a very valuable tool since it produces an up-to-date structured picture of the solution currently implemented. This picture may then be used to base a thorough evaluation of the status of the project, including any needs for manual redesign and reprogramming.

APPENDIX

The main characteristics of S-FORTRAN are:

(a) S-FORTRAN is a superset of FORTRAN (including the FORTRAN language extensions provided by various manufacturers).
(b) Any construct with a scope has both an opening and a closing delimiter. If the opening statement is XXX, the ending statement is of the form END XXX. (e.g., IF . . . END IF, DO WHILE . . . END DO WHILE).
(c) The IF includes any number of ELSEIF clauses and an optional ELSE clause. ELSEIF's are often convenient to prevent very deep indentation levels (and the so-called "wall to wall" ENDIF's).
(d) *Repetitive DO groups* include a DO FOR analogous to the FORTRAN DO loop, a DO WHILE, a DO UNTIL (which is in fact a DO AT LEAST ONCE UNTIL), and a DO FOREVER (an infinite loop).
(e) *Non repetitive DO groups* include a DO for bracketing statements, a powerful DO CASE, a DO CASE SIGN OF which is the equivalent of a three way arithmetic IF, and a DO LABEL to handle abnormal returns from subroutines and functions and end and error exits from I/O statements.
(f) UNDO is a mechanism to exit from a DO group prematurely. We have found this multilevel exit mechanism to be superior to introducing switch variables which tend to clutter the program and make its logic harder to follow. UNDO is applicable to any DO group, repetitive or not. It can be followed by a label if another DO group besides the innermost one is to be exited from.
(g) CYCLE is similar to UNDO but implies skipping any statement until the closing delimiter of a DO group is found. The test controlling the repeated execution of the DO group is then performed to determine whether to exit or repeat. CYCLE is only applicable to repetitive-DO groups.
(h) Internal parameterless procedures can be defined using PROCEDURE . . . END PROCEDURE. Their execution can only be triggered by an EXECUTE

(proc-name) statement. Premature termination of a procedure can be accomplished by an EXIT statement. Procedures share the same data space as the program in which they are contained.

REFERENCES

1. Ashcroft, E. A., Z. Manna, "The Translation of 'GOTO' Programs to 'WHILE' programs," *Proc. IFIP Congress 71,* Ljubljana, Aug. 1971.
2. Böhm, C., G. Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules," *Comm. ACM,* May 1966, pp. 366-371.
3. Caine, Stephen H., *Reference Guide to the XXX Language,* CFG 70-8-001, Feb. 1971.
4. Dijkstra, E., "GO TO Statements Considered Harmful," *Comm. ACM,* March 1968, pp. 147-148.
5. Donaldson, James R., "Structured Programming," *Datamation,* Dec. 1973, pp. 52-54.
6. Holmes, Charles E. and Leslie W. Miller, "Chief Programmer Experience," *GUIDE 37,* Nov. 1973.
7. Miller, E. F., *Extensions to FORTRAN and Structured Programming—An Experiment* General Research Corp., RM-1608, Feb. 1972.
8. Mills, Harlan D., "On the Development of Large Reliable Programs," *IEEE Symp. Computer Software Reliability,* 1973, pp. 155-159.
9. Peterson, W. W., T. Kasami and N. Tokura, "On the Capabilities of WHILE, REPEAT and EXIT Statements," *Comm. ACM,* Aug. 1973, pp. 503-512.
10. Wulf, William A., "A case against the GO TO," *SIGPLAN Notice,* Nov. 1972, pp. 63-69.

Toward improved review of software designs*

by PETER FREEMAN
University of California, Irvine
Irvine, California

INTRODUCTION

A good deal of effort has been invested in recent years to improve both the form of programs and the processes used to create them.[2,9,10] As the payoffs from this work become apparent and more widespread,[3] attention is turning to the form of designs and the processes used to create them.[1,5]

One aspect of software design, *reviewability,* has received scant attention. In this paper, we want to stress the importance of making designs reviewable and suggest an operational technique for aiding in their review. Underlying our discussions is the principle (perhaps obvious) that designs which can be easily reviewed have a better chance of meeting the expectations of their purchasers and users.

A methodology being developed by the author, *design rationalization,* provides a means for making software designs more reviewable before they are actually implemented. The body of this paper develops this idea.

Before we begin, three important points must be made. First, what we are proposing here is an *approach* to the improvement of design practice. It is not an algorithm. It cannot be applied to a situation without some thought and study. It certainly cannot be guaranteed to work in all situations. If you demand instant success, then look elsewhere! But, if you are concerned with improving design practice in your organization, especially with respect to reviewability, then we believe the idea presented here merits your study and experimentation.

Second, we must stress that there are already well-specified procedures for reviewing designs, but that in spite of their good intentions, they fail in some important respects. Much of the procurement of software by the government is now controlled by standards (promulgated by the Defense Department and other parts of the government) that spell out elaborate review procedures that must be carried out during the design phase (for example, Reference 11). Additionally, some organizations are experimenting with their own review standards aimed at improving the reviewability of software designs. (For example, good success has been informally reported by TRW at recent technical meetings with their usage of "unit development folders.") What these standards do not stress and what we consider essential to good review, is the recording of the reasoning behind design decisions, both local and global. It is this fact that our technique addresses most strongly and which we will stress in this paper.

The third point is that not all design situations are equal. We differentiate between *discovery design* and *routine design.* In the former, a great deal of creativity is required since the right structure (and even functions) for the software must be discovered during the course of the design. In the latter, the system being designed is similar to others which are well understood; thus routine design is more a process of choosing the right values for a set of parameters. Design of a program to prepare the payroll for an organization is clearly routine design. Design of a complex system to provide managers with real-time summary information automatically is discovery design, given our current understanding of such systems. In this paper we restrict ourselves to consideration of routine design situations.

We will describe and illustrate the design rationalization methodology and then show how it can be used to improve the reviewability of software designs. Because new methods are usually adopted slowly (and rightfully so), we close with some suggestions for experimentation with this technique.

DESIGN REVIEWS

Routine software designs are typically reviewed several times in different ways. Before we propose a way of improving design reviews, we want to consider some of their characteristics.

First, look at the range of review formats. An important part of the design process is a constant, but informal, review and iteration of the design by the designers themselves. When the preparation of a design is a large undertaking and/or is supported by a highly structured organization (such as the Federal government), formal design reviews are often specified (as in Reference 11) at which people other than the designers determine if the proposed design is acceptable (by whatever standards have been set up). Finally, the ultimate user of the

* This work was supported by National Science Foundation Grant GJ-36414.

software will review the design informally through usage and sometimes formally in preparation for requesting changes or a new design.

Our concern in this paper is the *reviewability* of a design—that is, the ease with which it can be compared to objectives. Designs cannot be executed directly as can programs, but it is still essential to compare them to desired criteria as early as possible in the development process.

While review of a design must necessarily mean different things to different people (depending on the methodology used, what is expected of the reviews, the stage of the process at which it is performed, and so on), let us be more explicit.

We see four possible components of any design review:

- checking for functional completeness;
- comparison of the design to operational goals and constraints;
- comparison of the design to non-operational goals and constraints;
- performance prediction.

"Operational" goals clearly and unambiguously spell out what is desired, while "non-operational" goals do not. For example,

Operational: "The system should provide a distinct error code for each error discovered."

Non-operational: "The system should handle errors cleanly."

It may be possible to characterize design reviews differently, but these four aspects capture most of what we see happening in the review of a software design.

The most prevalent question asked in a review is, "Will the system do what it is supposed to do?" The normal techniques of reading a design, perhaps aided by a structured walkthrough,[8] will generally suffice for answering this question. Because a design is typically stated in functional terms, most of it speaks directly to the question of what the system will do. While existing techniques do permit review for the completeness of major functions, it is still difficult to ascertain from a design whether small or unwanted functions are present.

Likewise, for explicitly stated structural goals or constraints, existing design formats permit at least a passable review. For example, if certain data structures or interfaces are part of the design requirements, then it is usually possible to determine if these requirements have been met by inspecting the design.

It is when we come to the last two components of a design review—comparison to non-operational requirements and performance prediction—that the need for improvement becomes most apparent. Even though it may be possible to make some design goals more operational (that is, detailed and open to objective evaluation), reviewers will still be asked to evaluate designs with respect to non-operational goals. For instance, consider the following design goals:

"The system should be tolerant of user mistakes."
"Only state-of-the-art techniques should be used."
"Output formats should be neat and readable."
"The system should be maintainable."

Determining whether these goals have been met or not requires the reviewer to interpret or infer information from the design and to provide a good deal of external information. Typically, the information provided in a design document is not the right type and/or is in the wrong form to permit such goals to be evaluated directly (if at all). The reviewer usually must proceed unaided.

Finally, designs provide almost no help at all for performance prediction. *Ad hoc* comparisons of parts of the design to previous designs (for which performance is known) may provide some meaningful predictions of resource usage. But even if the parts can be identified from the design, determining which are critical to performance and what the interactions between parts will be is very difficult. The information needed is simply not present in most designs.

Cutting across all aspects of design reviewing is the need for knowing the reasoning behind decisions. Rarely, if ever, in current practice does design documentation record the alternatives that were considered and the reasons for rejecting some of them. Yet, this information can greatly aid the reviewer in understanding the design and in evaluating it against stated objectives.

This brief look at the nature of design reviews certainly does not exhaust what can be said about them, but it should set the stage for considering how to improve them.

DESIGN RATIONALIZATION

There is no argument that software designs should be more reviewable and that if they were the resulting implementations could be improved. This is especially true in the case of routine designs where the form of the result is pretty well known in advance. We will outline below how design rationalization can be used to improve the

DESIGN PROBLEM 3: How should an error detected in a command string be handled?
  ALTERNATIVE 3-1: Abort the program when an error is found.
    EVALUATION 3-1: Easy to implement.
      Provides the user very little information.
      Wastes resources if the error occurs after much processing.
  ALTERNATIVE 3-2: Ask the user to re-enter the command string.
    EVALUATION 3-2: Must reset the state of the program.
      Makes the system "softer" on the user.
      Takes more processing time, even if no error encountered.
      May prevent waste of resources if trivial error.
  ALTERNATIVE 3-3: Try to correct the user's mistake.
    EVALUATION 3-3: Maximally useful to user.
      Substantial resources needed to try correction.
      Interaction with user more complex (must specify correction and allow override).
DECISION 3: Alternative 3-2, because it provides a balance between our goal to make the system easy on the user and the constraint that it be fast.

Figure 1—Information contained in a typical rationalization

reviewability of designs, especially in the area of recording the rationale for decisions.

The basis for design rationalization is the belief that designs can be improved by making them more rational. That is, design decisions should be based on logical reasoning, be supported by facts, and be recorded. The cornerstone of this technique is the *explicit* recording of design information in the form of design problems, alternative solutions, and the evaluations or arguments leading to the choice of a particular alternative.

The basic operation is the identification and recording of the information essential to a rational design. While variation in format is appropriate, the information shown in Figure 1 is fundamentally what goes into a rationalized design. The example shown there involves a single, rather low-level decision. In an actual rationalization we would record information pertaining to the entire design, both locally and globally.

If the information is collected and recorded as the design decisions are being made, we are doing a *synthesis rationalization.* If the information is primarily recorded after the design decisions are made, then we are doing an *analysis rationalization.* In either case, there are several important parameters: what features or decisions of the design shall be rationalized, how do we generate alternatives, what criteria shall be used for evaluating them, and on what basis should a decision be made.

Note that this methodology does not specify in what order decisions should be made. Neither does it spell out criteria for making decisions, except to specify that they should be made by considering alternatives and presenting evidence for and against each alternative. In this sense, design rationalization is more of a framework or forcing function within which particular decision strategies such as top-down or bottom-up can be used.

We must stress that the important aspect of design rationalization is its insistence on the *explicit* capture and recording of design information including the reasoning used. Without this, we have nothing but motherhoods about the importance of making rational decisions—which everyone already believes. With the explicit recording of information underlying decisions, however, we have a technique for increasing the rationality of designs.

With this brief introduction, let us look at how design rationalization can be used to improve the reviewability of designs.

A SCENARIO

The characteristics of a design review discussed above can be observed whenever a design is evaluated. For definiteness, though, let us focus here on the use of rationalization to improve formal reviews of routine designs.

Consider the following scenario:

1. *Initial specifications are prepared.* Assuming the specs are at a functional level, they are rationalized by providing explicit alternatives for critical specifications. Reasoning, based on facts is then spelled out for choosing a particular set of specifications.
2. *Specifications review.* The rationalizations permit the potential designers to understand more readily some of the specifications. They also permit those with funding responsibility to consider alternative forms of the system and to evaluate whether the system being specified is what is needed.
3. *Initial Design.* Once the specifications are finalized, an initial design is prepared, using the synthesis rationalization technique. Design decisions to be rationalized will include the overall organization of the system (control and data), choice of implementation language, choice of hardware, and other high-level decisions made at this stage. In addition, more detailed considerations of internal structure of the system may be documented by explicit lists of alternatives and evaluations.
4. *Internal Review.* When a design phase is completed, an internal review (such as the design walk throughs practiced by some organizations) and an analysis rationalization is performed by the designers and others in their immediate organization. This review may prompt changes to the design. The system's features are more thoroughly explained, additional alternatives are provided, and the evaluations of alternatives are strengthened. Some evaluations can only be made on the basis of global considerations after the entire design has taken shape. For example, choice of a data structure may depend on its usage by several different modules. This internal review has the effect of catching some design errors in-house while at the same time improving the explicitness of the rationalization.
5. *External review.* The initial design, augmented by the rationalization, is thoroughly reviewed by whatever outside agency has been designated to monitor progress. Rejected choices are explicitly spelled out in the rationalization along with the reasoning used to reject them. This permits reviewers to assess better the quality of the design and its fit to the specifications. Rejected alternatives may be recognized by the reviewers as important to some of the goals even though the designers felt they were not.
6. *Iteration and design refinement.* After any design review, changes to the design may be needed. After these have been made the design process continues by refining the design (or extending it, depending on the approach being used). Using the techniques outlined in 3, 4, and 5, the design and review iterative cycle will continue until a complete design ready for implementation is obtained.
7. *Implementation.* It is rare that the implementors of a system can proceed without making any changes or additions to a design. Typically, many decisions (hopefully, low-level) concerning the structure of the programs being built must still be made. The rationalizations now play a role in a different form of

design review. As the implementors seek to carry out the design, they must review it from the standpoint of understanding the intent of the designers when that is not clear and of making sure that decisions being made during implementation are not changing critical features of the design. The rationalization contains much of the reasoning information needed for this type of review.

8. *Redesign.* After a system is in use, more information is available on how well it fulfills its intended purpose. If the need for an improved system becomes clear then a major modification of the existing system or specification of a new system may become necessary. In this case, review of the design of the existing system to determine how it can be improved will be an important part of the design of the new system. This redesign process can profit from the rationalization by recovering rejected alternatives from the initial design.

This scenario illustrates the more important forms of formal design reviews often called for in the context of routine software design projects in large organizations. The use of design rationalization in these different review situations has been informally indicated, but the crux of the method—the explicit capture of design information otherwise lost and its presentation in a form convenient for review and comparison to goals—should be clear.

As a limited example of the usage of design rationalization, we performed a rather thorough analysis rationalization on a small system which had been designed and implemented as an improvement on an earlier system. The new system had several stated goals, including making it maintainable. The language used in the new system and some of the obvious features of the new design indicated that indeed this goal might have been achieved. The rationalization we produced, however, indicated that most of the effort in the redesign had been spent on local reorganizations of the system, with little thought given to overall control and code organization of the system. The careful analysis of the system that the rationalization supported convinced us that maintainability had been improved only marginally because of the lack of attention to overall structure. We were thus able to assess more accurately both the system and the techniques used to design it. This is illustrative of one type of benefit we would expect to reap from using design rationalization.

DISCUSSION

Our interest is in seeing design rationalization used to improve the practice of software design. To facilitate this, we will discuss some of its advantages and disadvantages in this section.

Advantages

One advantage comes from helping a reviewer identify alternatives. If one is knowledgeable in the area of the design, then alternatives may spring to mind easily. However, many reviewers will not be experts and will have difficulty knowing what alternatives (if any) might have been chosen for the design. Even for the expert, generating alternatives is often not easy.

The value of a rationalization in this respect is twofold. First, actual alternatives that have been rejected will be readily available for the reviewer to consider. Information that has been generated during the course of the design will not have been lost, but will be available for the reviewer. Perhaps even more importantly, the rationalization can serve as a pump-primer to get the reviewer started to thinking about feasible alternatives for the question at hand.

We are all familiar with the effect of being presented with a problem situation and of seeing at first only one solution. Unless one is familiar with the content area and has thought about the problem previously, it takes some effort to seek out alternatives. However, if someone suggests an alternative, even if it is not a good one, then we often can come up with additional suggestions much more easily. This is similar to one of the techniques suggested by De Bono[4] for facilitating lateral thinking—that is, of finding new ways of looking at an old situation.

Fundamentally, a reviewer is asked to certify that the decisions made by the designer are good decisions with respect to the goals and constraints of the design task. If the reviewer knows only the results of the designer's reasoning and not the steps by which the decisions were made, then the biases and knowledge limitations of the reviewer may seriously affect his or her judgments as to the quality of the design decisions. If the design is supported by explicit information in the form of a rationalization, however, then the reviewer can assess more easily the factual evidence and logical reasoning used by the designer.

This situation has an analogy in mathematics. If one is presented with a theorem, the truth or falsity of it may not be immediately evident and we may or may not be prepared to accept it as true. Given a step-by-step proof of the theorem, however, we can convince ourselves not only of its truth but also of why it is true (in terms of axioms and reasoning).

While the evaluations in a design rationalization are nowhere near as orderly as a mathematical proof, they do present the reasoning that has been used so that others can decide for themselves whether that reasoning is complete and valid. In addition, where the reasoning used to make a decision involves assumptions, exposing this reasoning will permit the reviewers to discover and assess the validity of the assumptions (since many of them may be related to the user environment which the reviewer knows more about anyway).

The advantages discussed here have touched on what we believe to be the basic advantage of design rationalization for improving reviewability: It forces the explicit recording of decision reasoning information which is otherwise lost.

Disadvantages

It should be clear that producing a rationalized design will, in general, take more effort than producing an rationalized design. At a minimum, the effort needed to record alternatives and evaluations, even if otherwise generated, is added effort. However, we have typically found that generating the explicit lists of alternatives and evaluations also requires a good deal of effort. Finding meaningful evaluations, that relate the decision under consideration to the goals and constraints of the problem, is often difficult in the absence of an underlying theoretical basis or quantitative evaluation technique. The advantages of design rationalization (both for reviewability and quality of a design) must be weighed against this added cost.

At present we cannot offer explicit suggestions for choosing the design problems/features to rationalize nor surefire methods of generating alternatives and evaluations. It is clear that one cannot rationalize every single decision in a program of any significant size. Further, important decisions in one design may have no major role in another. Choosing the important ones (those for which the choice of a solution has some definite effect on the resulting design or its use) is difficult.

A more subtle problem that may not be immediately evident concerns the level at which decisions are made. The thrust of design rationalization as we have presented it above is to make decisions at a local level. Basically, the question asked is, "What is the best alternative, and why, for this particular decision?" This leads to local optimization, which in many cases will not be optimal. That is, sometimes we must make decisions taking into account the alternatives for other decisions which have not yet been resolved. It is to help alleviate this concentration on the local context that we have suggested in the scenario above that some rationalization be done after the initial design is completed.

Research

We are continuing our investigations of design rationalization both to provide additional evidence of its advantages and to find ways of reducing its disadvantages. Included in this work are some informal investigations of design situations in which some designers use rationalization and others do not, development of techniques to make easier the choice of problems and generation of alternatives and evaluations, and the construction of tools to help in recording the information. These studies are described in References 6 and 7 and other working papers available from the author.

SOME SUGGESTIONS FOR USING DESIGN RATIONALIZATION

Any methodology not based on formal techniques is open to interpretation by those using it. Such interpretations are, in fact, required in most cases to make methodology useful in the context of a particular organization or a particular type of task.

Thus, we understand that design rationalization must be adapted to your particular organizational task context. While it is difficult to predict the difficulties you will encounter, our limited experience with helping others use it does suggest some guidelines.

Remembering that we are concerned here with formal reviews of routine designs, we suggest the following:

1. Choose a small, but realistic design problem on which to try out the technique. Make sure the design is of a system of which the designer has some knowledge.
2. Use a design project that is "real" (not done just for experimentation).
3. Make sure sufficient time and resources are allocated to the project so that the designers are not under pressure.
4. Use at least two designers (but probably not more than three) so that they can work as a group when generating alternatives and evaluations.
5. Use your normal design techniques augmented by the use of rationalizations as suggested in the scenario above.
6. Carefully choose some criteria by which you can judge whether the rationalization assists in design review. Some suggestions are: number of design flaws discovered relative to similar projects, level of detail of design flaws discovered, perceived ease of review by reviewers, time taken to review design documents.
7. Maintain careful observations of the use of rationalization to permit later analysis of the trial.
8. Assess the trial when completed. If rationalization seems to help in your situation, even a little, try to find ways to improve the technique for your situation.
9. Try it again.

These suggestions can be boiled down to a simple statement: Approach the use of design rationalization from the standpoint of an experimental fitting of an idea to your situation and expect to make changes.

CONCLUSION

Development of techniques for the review of software designs has been largely neglected. We have described a methodology, design rationalization, which has characteristics that will help improve the reviewability of designs. We have given a scenario for its usage and discussed some of its advantages and disadvantages. Suggested guidelines for trying it out were given.

We have stressed that the strength of design rationalization lies in its forcing *explicit* recording of design information, especially that which explains the reasons behind features of the design. The existence of this information in a form that permits independent review of design choices and the reasoning leading up to them should assist in most situations.

We have not spelled out an explicit technique for one to follow. Rather, we have described an idea, assessed its use for improving the current practice of design review, and suggested ways in which it can be adapted to varied organizational settings. As with much of software engineering today, the application of this idea in large-scale situations must ultimately be carried out by those with software creation problems to solve.

While our research continues into the ramifications of this idea and the techniques for using it, others can profit from trying it in their contexts. We recognize the difficulty of changing one's patterns of doing something and the difficulties in forcing oneself into the discipline of design rationalization. Yet, only through trial and error usage of this and other proposed methodologies can we gradually develop the tools necessary for the routine design of large and important classes of software.

REFERENCES

1. Brown, R. R., "1974 Lake Arrowhead Workshop on Structured Programming," *Computer,* October, 1974, pp. 61-63.
2. Dahl, O. J., E. W. Dijkstra, and C. A. R. Hoare, *Structured Programming,* Academic Press, 1973.
3. *Datamation,* special issue on structured programming, December 1973.
4. De Bono, Edward, *New Think,* Basic Books, 1972.
5. Freeman, Peter, "Automating Software Design," *Computer,* April 1974.
6. Freeman, Peter, *Reliable Software Through Rational Design,* ICS Technical Report #55, University of California, Irvine, October 1974.
7. Freeman, Peter, *Design Rationalization,* ICS Technical Report #57, University of California, Irvine, November 1974.
8. IBM, *Structured Walkthroughs,* training brochure.
9. Mills, H. D., "Top Down Programming in Large Systems," in *Debugging Techniques in Large Systems,* R. Rustin (ed.), Prentice-Hall, 1971.
10. Parnas, D. L., "On the Criteria to the Used in Decomposing Systems into Modules," *Comm. ACM,* December 1972.
11. U.S. Air Force MIL-STD 1521, *Technical Reviews and Audits for Systems Engineering and Computer Programming,* available from National Technical Information Service, Springfield, VA., September 1972.

Understanding software through empirical reliability analysis*

by T. A. THAYER
TRW Systems Group
Redondo Beach, California

INTRODUCTION

The need for improving the reliability of delivered software is becoming increasingly obvious to both the purchasers and producers of today's software systems. As noted by Boehm,[1] the records show many examples of software systems which, when delivered for operational use, either performed in a degraded fashion or failed to perform at all. The results are higher software costs and delays in operational usage.

Purchasers of software are demanding more reliability but are not presently sure how to specify what they want in the form of written requirements. Producers of software, in an effort to provide this increased reliability, are imposing new controls on the software development process. These new controls are both managerial (chief programmer teams, independent test and audit groups, etc.) and technological in nature (structured programming, coding standards, test tools, etc.). Although these controls are intuitively beneficial, little hard data exist to support claims of benefit. There is even less information on the benefits of these controls in specific software applications; e.g., batch versus real time software, OS versus applications software, programs written in assembly language versus higher order language.

Assuming that information is collected and available, what can we expect to gain from its analysis? The long range benefits are the most obvious. We will learn, for example, how large a software module can be without affecting understandability and how small it can be before partitioning problems are encountered. Specific tools and techniques to improve the development process will also be identified. These are all things that can be applied in the future, to the next project. But, what about the near term payoff for the project supplying the data? A project manager, when asked to contribute to data collection and analysis activities, will invariably ask if there will be a benefit to his project. The benefit to the on-going project comes from increased awareness of problems and better control over the development process, both contributory to production of more reliable software. Adequate data on software reliability can thus aid both current and future development projects—if methods for analyzing and understanding these data are developed.

In a study* being performed by TRW for the Rome Air Development Center, data from four large software systems are being analyzed to determine the types of errors found in software during testing. The objective is principally to recommend new development or test techniques for the detection and prevention of software errors, but we are also attempting to model software reliability. In the course of supplying real** data descriptive of software reliability and for model evaluation, we have had to determine (1) what data are generally available, (2) methods for collecting and storing these data, (3) methods for describing software errors, (4) methods for characterizing the software, and the development and test processes in quantitative terms, and finally (5) methods of analysis. Although the projects studied have varied greatly in size, language, operating mode, and structure, the data available during the development process were similar for each project: error data, recorded in various forms of software problem reports (SPR) and ancillary project data needed to understand and support analysis of the error data. Although the data were not generated specifically for the study, we found that we could do much to quantify software reliability and the characteristics of the software itself, as well as improving our understanding of both the software and the development process. Some results of the Software Reliability Study will be presented to illustrate the benefits of software reliability data collection and analysis. Also presented are some recommendations for identifying data that need collecting.

SOFTWARE RELIABILITY

As the reader will note, the term "software reliability" has been used several times in the foregoing paragraphs without definition. Although the term is defined in a number of references,[2,3] generally in conjunction with modeling work, the following definition is taken and offered as a background for the empirical approach to characterizing reliable software:

> Software possesses reliability to the extent that it can be expected to perform its intended functions satisfactorily.[4]

* Portions of the work presented in this paper are sponsored by Rome Air Development Center under Air Force Contract F30602-74-C-0036.

* The Software Reliability Study.
** A key study feature.

CATEGORY ID	CATEGORIES	PROJECT 2					PROJECT 3	PROJECT 4
		MOD1A	MOD1B	MOD1BR	MOD2	TOTAL		
AA000	COMPUTATIONAL ERRORS	0	0	0	0	0	0	0
AA010	Total number of entries computed incorrectly	0	0	0	0	0	19	0
AA020	Physical or logical entry number computed incorrectly	8	6	2	21	37	27	0
AA030	Index computation error	2	7	1	17	27	31	4
AA040	Wrong equation or convention used	3	6	4	11	24	57	0
AA041	Mathematical modeling problem	0	0	0	1	1	7	0
AA050	Results of arithmetic calculation inaccurate/not as expected	0	0	2	5	7	74	0
AA060	Mixed mode arithmetic error	0	0	0	0	0	0	2
AA070	Time calculation error	2	1	5	13	21	36	0
AA071	Time conversion error	0	0	0	0	0	7	0
AA072	Time truncation/rounding error	1	0	1	2	4	2	0
AA080	Sign convention error	0	2	0	5	7	16	0
AA090	Units conversion error	1	0	2	15	18	28	1
AA100	Vector calculation error	1	0	0	0	1	13	0
AA110	Calculation fails to converge	0	0	3	2	5	4	0
AA120	Quantization/truncation error	1	4	1	4	10	32	0
	TOTALS	19	26	21	96	162	353	7
BB000	LOGIC ERRORS	0	0	0	0	0	0	0
BB010	Limit determination error	2	5	4	5	16	37	1
BB020	Wrong logic branch taken	1	4	1	5	11	49	0
BB030	Loop exited on wrong cycle	0	0	0	0	0	0	2
BB040	Incomplete processing	4	2	4	10	20	58	0
BB050	Endless loop during routine operation	1	4	1	0	6	35	0
BB060	Missing logic or condition test	6	9	8	26	49	233	72
BB061	Index not checked	2	0	0	1	3	59	0
BB062	Flag or specific data value not tested	5	4	8	34	51	139	0
BB070	Incorrect logic	0	0	0	0	0	0	57
BB080	Sequence of activities wrong	4	7	2	18	31	57	3
BB090	Filtering error	1	3	0	4	8	7	1
BB100	Status check/propogation error	6	3	1	2	12	103	0
BB110	Iteration step size incorrectly determined	0	0	0	0	0	0	1
BB120	Logical code produced wrong results	3	4	1	19	27	39	0
BB130	Logic on wrong routine	0	0	0	2	2	6	0
BB140	Physical characteristics of problem to be solved, overlooked, or misunderstood	1	1	0	0	2	64	2
BB150	Logic needlessly complex	0	0	0	0	0	5	0
BB160	Inefficient logic	0	2	0	2	4	26	1
BB170	Excessive logic	1	3	1	9	14	18	0
BB180	Storage reference error (software problem)	0	0	0	0	0	2	0
	TOTALS	37	51	31	137	256	937	140

Figure 1—Sample error category list

Assuming that documented errors represent an inability to perform intended functions satisfactorily, error-free software would be reliable software. This assumption is one that is made here.

In the definition above, the word "satisfactorily" also begs definition. A universally applicable quantitative definition does not presently exist; however, in the world of software problem reports mere creation of the report registers some dissatisfaction with performance, whether the report documents a real problem, a need for software enhancement, or what turns out to be no problem at all.

Using the definition given above, the number of errors documented on problem reports can be used as an indicator of software reliability. However, raw counts of errors don't tell the whole story and can be misleading. Therefore, further analysis is necessary to determine the types of errors, when and where they were introduced, how they were detected, and their impact on the operation of the software system.

It is possible to categorize software problem reports symptomatically, at the user level, and according to cause, at the code level. Symptomatic categories describe the failure, and causative categories describe the error. These are often quite different. Attempts to establish software error category lists for multi-project applications have met with some success. The principal findings being that errors are repeated and tend to fall into common categories, that lists have to provide a fair amount of detail to yield useful information, and that they also have to be short and comprehensive so that the fixer* of the problem can be encouraged to provide the analysis. A portion of an error category list is given in Figure 1 along with empirical data from three software projects. This list presents data generated during development of a new software system (Project 3), during development of four updates to an operational system (Project 2), and during operational usage (Project 4).

SOFTWARE CHARACTERIZATION

In order to better understand why elements of the software have high or low error frequencies, it is necessary to collect data on the software itself. Two types of software characteristics were defined in the Software Reliability Study, structural and subjective characteristics. Structural characteristics are those that can be obtained from the source code, where the errors are ultimately found. These character-

* He alone is close enough to the actual error to assign a causative error category and he must make the assignment while the solution is fresh in his mind.

TABLE I—Routine Structural Characteristics

- Routine size (source statements and machine instructions)
- Number of branches
- Branch statements and nesting levels
- Entry/exit points
- Routine interfaces
- Data base interfaces
- Calling sequence arguments
- Code type
  - —computational
  - —data handling
  - —logical
  - —I/O
- Comments

istics (a typical list is presented in Table I) can (or should) be collected automatically at the routine level with what is commonly called a static language analyzer.

Subjective characteristics are those that are obtained from the people who produced the software; the programmers, the design engineers, the testers, and their managers. Although methods for quantifying these subjective characteristics are not well established, this information can add insight to the error history data that is not evidenced by the more accurately measured structural characteristics. However simple a routine might be structurally, if it was *felt* that the routine was (or would be) a problem for some other reason, the error history can be affected by this feeling. A typical example concerns the software that has been produced previously for another project. The initial feeling is that it should be an easy task to convert an existing routine for use in a similar (but not identical) software system. This situation invariably receives less attention than the newer software and just as invariably represents more work than was originally conceived. On the other hand, portions of the software thought to represent potential problems are given much attention, and this can be seen in the error history data.

Subjective characteristics dealt with in the Software Reliability Study were difficulty and software type. Difficulty was assigned to each of five development disciplines: design, code, debug/test, implementation, and documentation. There were five routine types* considered, too: executive or control, setup, computational, data handling, and input or output. These data were very helpful in understanding error histories.

Attributes and metrics

Attributes identify specific software characteristics. These might involve such things as difficulty, complexity, and readability. Metrics quantify attributes. They can range from simple counts of source statements, logic branches, and routine interfaces to calculated values of difficulty, complexity, test sufficiency, and readability, these latter calculated metrics requiring mathematical expressions of definition. Metrics are being used in understanding and explaining error histories, and it is hoped that they will eventually provide means for comparing varying and unrelated software projects. Table II presents several examples of metrics considered in the Software Reliability Study.

TABLE II—Sample Metrics*

Logical Complexity

$$M_{\text{LOG}_1} = \frac{LS}{TS}$$

where: LS = Number of logical (branch creating) statements
TS = Total statements in routine

Logical Statement Complexity

$$M_{\text{LOG}_2} = \frac{TB}{LS}$$

where: TB = Total branches in the rountine
LS = Number of logical statements

Loop Nesting Complexity**

$$M_{\text{LOOP}} = \sum m_i W_i$$

where: i = nesting level (1st is non-nested)
m = number of loops

$$W_i = \frac{4^{(i-1)}}{\sum_1^m 4^{(i-1)}}$$

* Note that these metrics are, by design, relatively simple and incorporate parameters which are thought to be available from almost any software system.

** This particular form of the equation resulted from a combination of intuition about loop complexity and evaluation with real data where nesting reached the tenth level in some instances.

Of course, the definition of such things as complexity is judgmental on the part of the analyst and not altogether independent of what data are available. Some metrics have shown nothing when an attempt is made to correlate them with error frequencies; others can't be supported with real data for lack of collection tools. We are forced to admit that evaluation of metrics is part of the study process, and it will take several iterations on a number of differing software systems to identify the more universally applicable metrics. One point that should be made, however, is that every attempt to define metrics, even if unsuccessful, adds to the analyst's understanding of the software under study. Every attempt to understand what makes software complex, more readable, easier to test, or easier to maintain aids in the understanding of software errors and provides a potential for uncovering improved development or test techniques.

RESULTS FROM SOFTWARE RELIABILITY ANALYSIS

Four examples are given below to indicate ways in which understanding of software and the development process can be improved through analysis of empirical reliability data.

* This was considered a subjective characteristic because routines labeled as one type by a programmer often were characterized structurally by a different type of code (e.g., I/O routines tended to be predominantly made up of logical code).

Detecting on-going problems

During the formal test period for one of the projects examined in the Software Reliability Study, it was discovered that a portion of the software system was judged to be very error prone. Using a simple count of errors discovered as a guage this software looked no worse than other software in the system. However, by also considering the criticality and type of the relatively few errors being encountered, the system engineering group would have been able to identify a problem traceable to rapidly changing requirements and the need for substantial redesign and test prior to delivery. The symptoms of the problem were present throughout the development cycle (even as early as preliminary design), but there was no effort to create an historical dossier of all available data and tie it to the offending software. As a result, the severity of the problem was not realized and the software continued to be a problem even after delivery.

Benefits of new software development techniques

One of the projects analyzed in the Software Reliability Study utilized a number of advanced development and test techniques and tools. Among these were structured programming, a 100 executable statement limit on routine size, programming standards enforced by audit tools, and rigorous development testing requirements, including a requirement to execute all code at the routine level. Test tools include a dynamic test monitor to track the amount of code exercised. Preliminary results* of a comparison of this and other projects show a tendency for errors to be found sooner and for the total error count to be lower. Figure 2 depicts an actual cumulative error history (solid curve) of a project using only standard development techniques and the suspected results of using new development and test techniques (dashed curve).

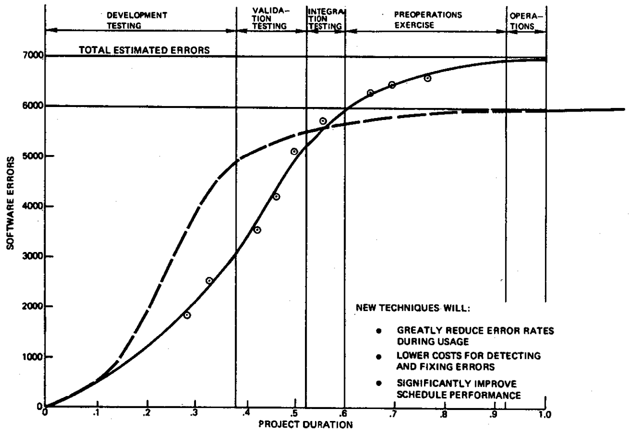


Figure 2—Benefits of new development techniques

Other preliminary results of error analysis show a lower percentage of the total data processing errors* using newer techniques, presumably because of the smaller, better structured, and easier to test routines. On the other hand, the percentage of interface errors is greater for the system with many smaller routines.

Understanding error rates

Hopefully, a few (ideally one) independent variables correlate well with the number of errors in the software and enable us to use the independent variables values to understand ultimate reliability. Our approach has been to use a large sample of real instances to compute the correlation between metrics characterizing the software and the number of errors in the software. The metric that correlated best with the occurrence of actual software errors was the simple measure of size in source statements. (Routine complexity expressed as a count of logic branches showed similar trends.)

Although correlation for all routines in the software system taken together was poor (correlation coefficient = 0.304), selected groupings of routines within the system, specifically routines which function together to form parts of each software subsystem, correlated very well. The highest correlation coefficient encountered was 0.910 for a straight line curve fit of 25 routines ranging in size from 10 to 2300 source statements, but the bulk of the correlation coefficients were on the order of 0.7. Although slopes varied from group to group and the best fit was not always linear, a typical data point was 20 errors/1000 source statements.

Results from other metric analyses, aside from providing insight on what makes software complex and how to measure this complexity, tend to show that such things as schedule pressures, the availability of needed resources, and requirements have as much or more to do with the eventual quality of the software as do the structural characteristics of the system. For instance, routines which exhibited similar error histories but were quite different in structure and software type were subject to the same external influences (e.g., they were produced by the same developers, suffered from poorly defined or rapidly changing requirements, lacked realistic data base values for testing, etc.). Unfortunately, these influences are not easily quantifiable through present techniques, but their effects are seen in the data.

Error sources

One of the most enlightening results of the Software Reliability Study has centered on where errors were introduced. Although all errors detected past the design phase are found in the code during some form of checkout or testing, these errors need not originate from the coding phase. That is, they may be due to design or even requirements definition errors. For one of the systems analyzed in the Software Reliability Study (a command and control system) accurate

* Results are considered preliminary because, at the time of writing, the project was still under development and the total error history was not complete. We eagerly await completion of this project when our hypothesis testing can be completed.

* For example, iterative procedure, bit manipulation, and indexing errors.

TABLE III—Relative Frequency of Design and Coding Errors

Modification	No. of Source Statements in Modification	Total Errors Encountered	% Design Errors	% Coding Errors
A	1253	152	73.6	26.4
B	9880	156	73.7	26.3
C	779	73	35.6	64.4
D	9631	419	51.6	48.4
E	4575	199	58.8	41.2
F	—*	113	61.9	38.1
G	—*	120	65.8	34.2

* Size data not accurate.

data was available which identified a failure as due to either a design or coding error. No data were available to identify those errors stemming from poor requirements, but these errors were considered to be negligible. Data accumulated over a three year period in seven modifications to the system is summarized in Table III.

This preponderance of design errors is supported by data from the three other systems also, although percentage breakdowns are not so easily obtainable due to inaccuracies in the data collection process. Among the more common errors were those where the design did not take into account certain aspects of the physical problem to be solved. Software failures, in this case, manifested themselves in a number of ways; the error was in the design in the form of missing logic or condition tests. In one system analyzed, where approximately 40 percent of the budget was spent on design, 9.7 percent of a total of 4439 reported errors were of the type described above (i.e., the approved design was missing some necessary logic). Two other systems exhibited 6.8 percent (out of a total of 1498) and 13.4 percent (out of a total of 539) for the same type of error. Typically it is this type of error, a fundamental design error, that is most difficult (and costly) to fix. The cost involved in diagnosing and correcting design and coding errors has been quantified by Shooman and Bolsky.[5] In their paper the effort in manhours to diagnose and correct design and coding errors were compared. Results showed that design errors required an average of 3.1 manhours to diagnose and 4.0 manhours to correct. Coding errors required an average of 2.2 manhours to diagnose and 0.8 manhours to correct.

With design errors being more costly and more frequent than coding errors, the conclusion is that improvements in design technique and design aids or tools are needed. The coder has the compiler to help him, and the tester, in recent years, has had automated test tools. The designer presently has only his wits to help him.

NEEDED DATA

Key to the empirical approach of determining software reliability is the existance of good (well organized and defined) data. Presently data collection schemes don't provide enough of the right kind of data, nor are the data collected at the right time.

Figure 3 illustrates a typical software development project by phase and points out when various types of data can become available. Ovals indicate error or problem data that can be produced and collected. Ancillary data, shown below the phases of the development cycle, become available roughly at the times indicated, are absolutely necessary to the understanding of the error histories, and can be extremely useful in assessing on-going project performance. Triangles along the base of Figure 3 denote points at which snapshots of the software structural characteristics can be taken to gain a picture of the volume of change.

Individual data items will vary from project to project, and most projects have a potential for creating a tremendous amount of data. From our work in collecting and analyzing these data, three general categories appear to be most important in answering both reliability and status related questions. These, along with needed data within each category, are given below.

Error data

Information concerning errors is generated throughout the development cycle of the project. No matter which phase in the cycle provides this information, data should be recorded on collection forms tailored to the data and should be collected as near the identification date as possible. Furthermore, categorization of errors by type should be performed by the problem fixer, if at all possible. Questions that should be addressed in the collection process are the following:

- When was the error introduced? Was the error made during requirements definition, software design, coding, testing or operational maintenance?
- How critical to successful operation of the software system is the error? This amounts to a priority on the need

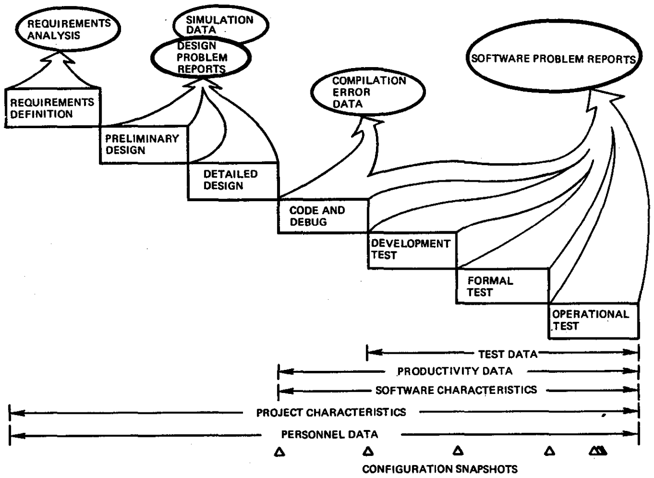


Figure 3—Software development and reliability data

for corrective action and is useful as a normalizing factor in comparing data from different projects.

- How and when was the error found? This is useful in evaluating the software development and test processes.
- What test "stress" was applied to produce the failure? This is important to answering the question "How much testing was done?"
- Was the error independent of other errors or the result of a previous fix? This information is important in assessing the quality of software maintenance activities.
- What was the resource expenditure in manhours and machine time to diagnose and correct an error?

Characteristics of testing

It is testing that produces the bulk of the software problem reports which serve as the basis for analysis. The number of problems documented will depend to a great degree on the formality of testing and when this formality is initiated. Regardless of when this control is imposed, it is important to be able to relate software failure data to specific dynamic executions of specific test procedures. If these test procedures are then evaluated according to the amount of stress they place on the software, an attempt can be made to determine when enough testing has been done and when the quality of the software is resulting from a particular test program. Measures of stress vary from CPU time per test execution to the number of program segments exercised and the representativeness of the input data base, and the worth of any of these measures depends upon who you talk to. The important point here is to know how much testing, according to some measure, produced a particular error history. Test stress has been and remains to be difficult data to collect. Both the tools and the methodology for collection are in their infancy, but recent work in software reliability modeling offers promise in this area.[6-8]

Characteristics of the development cycle

These characteristics will vary from project to project but can generally be grouped according to schedules, resources, and personnel. Although important to the understanding of error trends and reliability related data, it is the characteristics of the development cycle itself that often point to problems that are documented on error reports later in the project. Project resource availability can be a powerful tool in identifying problems. For example, unavailability of computer time during the compilation and debug phase may, to meet schedules, force reduction of the amount of detailed development testing a critical routine undergoes prior to formal testing or delivery to a customer. Early identification of this situation could allow additional preliminary testing steps to be taken prior to beginning of the more formal testing involving that routine. Parameters useful in characterizing the development cycle are given in Table IV below.

TABLE IV—Development Cycle Characteristics

- Resource Availability
  - —Machine time
  - —Software tools
  - —Externally acquired resources
- Schedule Data
  - —Behind, on, or ahead throughout the development cycle
  - —Planning
- Personnel Data
  - —Number of programmers
  - —Load factor on each programmer
  - —Supervisor's rating of programmer

CONCLUSION

Data collection and analysis represent an added workload for the project providing the data; however, the yield of useful information for both technical and management control will be increased. Done properly, manpower would be provided for collection of data throughout the life of the project; much of the useful data is perishable and must be collected as it is created. Further project involvement is required in the analysis of the data because individual performers alone are able to provide some of the data with sufficient accuracy (e.g., causative error data). Access to project experience is essential to accurate interpretation, especially if the analysis is done independently by other than project performers.

Collecting and analyzing data makes it possible to answer questions concerning software reliability with something other than philosophy and speculation. Increased awareness of the types of errors encountered, the characteristics of the development process that produced them, and the conditions under which they occurred is the first step in quantitatively specifying measures of quality to be used by purchasers and producers of software alike.

BIBLIOGRAPHY

1. Boehm, B. W., "Software and Its Impact: A Quantitative Assessment," *Datamation*, Volume 19, No. 5, May 1973.
2. Shick, G. J. and R. W. Wolverton, *Assessment of Software Reliability*, TRW Software Series Report TRW-SS-72-04, September 1972.
3. Wagoner, W. L., *The Final Report on a Software Reliability Measurement Study*, Aerospace Report No. TOR-0074 (4112)-1, 15 August 1973.
4. Merritt, M. J., et. al., *Characteristics of Software Quality*, TRW Software Series Report TRW-SS-73-09, 28 December 1973.
5. Shooman, M. L., and M. I. Bolsky, "Software Errors: Types, Distribution, Test, and Correction Times," *Proceedings of 1975 International Conference on Software Reliability*, April 1975.
6. Shooman, M. L., "Operational Testing and Software Reliability Estimation During Program Development," *Proceedings of IEEE Symposium on Computer Software Reliability*, 1973.
7. Jelinski, Z. and P. B. Moranda, "Applications of a Probability-Based Model to a Code Reading Experiment," *Proceedings of IEEE Symposium on Computer Software Reliability*, 1973.
8. Nelson, E. C., *A Statistical Basis for Software Reliability Assessment*, TRW Software Series Report TRW-SS-73-03, March 1973.
9. Clapp, J. A. and J. E. Sullivan, "Automated Monitoring of

Software Quality," 1974, *AFIPS Conference Proceedings*, pp. 337-342.

10. Brown, J. R., et. al., *The Quantitative Measure of Software Safety and Reliability*, TRW Software Series Report TRW-SS-73-06, 24 August 1973.

11. Amory, W. and J. A. Clapp, *A Software Error Classification Methodology*, MITRE Technical Report No. MTR-2648, Volume VII, 30 June 1973.

12. Youngs, E. A., *Error-Proneness in Programming*, Doctoral Thesis in Computer Science, University of North Carolina 1970.

Dynamic dispatching in job class scheduled systems*

by JON C. STRAUSS
University of Pennsylvania
Philadelphia, Pennsylvania

INTRODUCTION

Many different approaches have been attempted to the problem of controlling the processing of a multiprogramming computer system so as to improve performance. In Reference 1, Bowdon, et al.. simulate an IBM 360/75 under OS with HASP and adjust job initiation priority based on estimates of processing time to effect a shortest-processing-time discipline. Their simulation demonstrates a dramatic increase in thruput and decrease in turnaround as a result of this job scheduling modification.

In Reference 2, Wulf avoids potential problems of scheduling based on poor a priori knowledge of job characteristics by dynamically controlling individual job execution based on measured use of system resources by the jobs. In attempting to balance system resource utilization however, Wulf's controller removes jobs from competition for system resources in favor of other jobs that better contribute to current system balance. In so doing the controller may be obtaining good current performance, but because this alteration is done independent of the relative number of jobs of different types waiting to be processed poor overall performance could result. Northouse and Fu[3] describe a scheduling system that automatically classifies jobs based on resource requirements and then runs combinations of jobs from the classes to maximize a performance index subject to meeting preset constraints on numbers of jobs run. Their approach, like that of Bowdon, et al.,[1] appears to be sensitive to poor information on job characteristics, but their problem definition admits to optimizing performance in terms of the presented workload.

One other work on scheduling is relevant to the thrust of this paper. In Reference 4, Brinch Hansen develops an interesting compromise between the attractive short job response of shortest-processing-time scheduling and the good long job response of first-come-first-served (FCFS) scheduling. This alternative, response ratio scheduling, schedules the job for next initiation that has the highest ratio of response time to service time. This scheduling clearly favors short jobs, but it also limits the waiting time of longer jobs.

The referenced works[1–4] are not oriented toward the job class scheduling characteristic of IBM OS systems. Classes of jobs are automatically identified in Reference 3 and Reference 1 deals with the IBM environment, but neither of these papers makes use of the additional information concerning job characteristics that presumably is present in some, if not all, job class scheduled systems. For those systems where user jobs are placed in classes on the basis of type of user (e.g., academic, administrative, etc.) or characteristic resource demands (e.g., multi tape jobs), or both, knowledge of the characteristics of the currently processing job in a class provides a good first approximation of the characteristics of other jobs in that class.

Typically in a job class system, jobs are scheduled for initiation from within classes and in many systems jobs are multiprogrammed one from each class. This paper restricts its scope to such systems and investigates algorithms for dynamically altering the relative preemptive CPU priority of the different classes so as to improve overall system performance. No attention is paid to the response time of individual jobs within classes (it would be possible for example to impose shortest-processing-time job initiation scheduling within a class and still have the general properties presented here).

A recent paper by the author and A. Chiang[5] defines and solves a maximum thruput problem for an idealized job class system similar to OS/360. A dynamic control algorithm is developed that maximizes average system thruput while providing equal average turnaround to the different job classes. The control algorithm is shown to be a variation of the HASP Execution Task Monitor (HETM) dynamic dispatching algorithm[6] the performance effects of which are investigated in Reference 7.

The most important contribution of Reference 5 is not the specific solution to the defined problem. Rather, the paper stresses two main points: (1) the performance of a system must be defined in terms of the work to be done, and (2) the desired system operation should be specified explicitly in terms of performance and not in terms of a means of obtaining performance.

The next section reviews the properties of the HASP dynamic dispatching algorithm, presents the equal job class turnaround algorithm, and develops the equal job class response ratio algorithm. A simple two job class problem is then solved analytically and the performance of the system for four dispatching algorithms (class priority, HETM, equal

* This work was performed at Washington University, St. Louis Mo. under support of NSF Grant GJ-33764X.

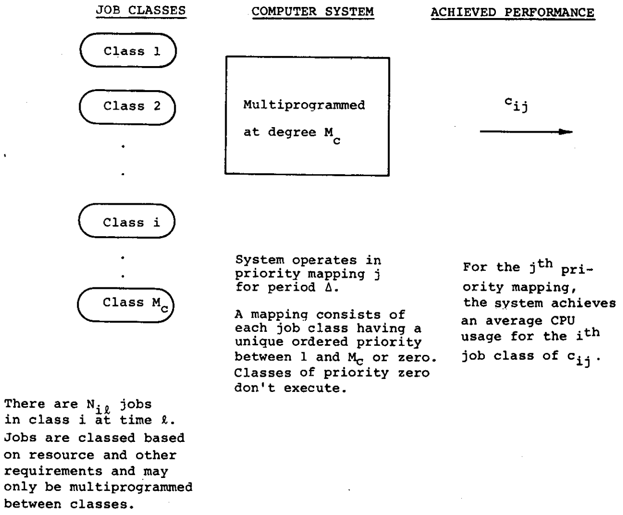

JOB CLASSES
COMPUTER SYSTEM
ACHIEVED PERFORMANCE
Class 1
Class 2
Multiprogrammed at degree M_c
c_{ij}
Class i
Class M_c
System operates in priority mapping j for period Δ.
A mapping consists of each job class having a unique ordered priority between 1 and M_c or zero. Classes of priority zero don't execute.
For the jth priority mapping, the system achieves an average CPU usage for the ith job class of c_{ij}.
There are $N_{i\ell}$ jobs in class i at time ℓ. Jobs are classed based on resource and other requirements and may only be multiprogrammed between classes.

Figure 1—Pictorial representation of job class orientea computer system

job class turnaround, and equal job class response ratio) is compared for different system loading conditions. The Conclusions section summarizes the main results.

THREE DYNAMIC DISPATCHING ALGORITHMS

The HETM and equal average job class turnaround dynamic dispatching algorithms are reviewed and the equal job class response ratio algorithm is developed.

The job class scheduled computer system of concern is depicted in Figure 1. Jobs are multiprogrammed one from each of M_c job classes. Jobs from class i require an average C_i units of CPU time to complete. At time $t_l(=l\Delta)$, there are N_{il} jobs in the ith input job class (queue). Jobs are classed on the basis of resource requirements and it is assumed initially that the classes are equally worthy of system attention. (External specification of priority may be used to order jobs within classes, but not the relative worth of one class with respect to another.)

The system control mechanism, which is similar to that available in OS/360, involves the specification of the unique internal preemptive *CPU* priority to be given to a single task from each of the M_c classes over a control interval Δ. This priority ordering is specified every Δ time units on the basis of the observed performance of the system. The specification of the ordered priorities of the M_c classes is termed a priority mapping. There are M_p priority mappings. The system behavior is observed through the job *CPU* usage c_{ij} attained for a task from the ith job class when it is multiprogrammed in priority mapping j.

In a standard OS/360 job class scheduled system, there is no dynamic dispatching. A fixed relationship exists between the job class and the preemptive *CPU* dispatching priority of tasks from jobs in that class. (This statement must be qualified to the extent that dispatching priority goes with the partition or region and several classes may feed the same partition or region). The intent of this fixed dispatching scheme is to allow the installation to class jobs based on knowledge of *CPU/I-O* characteristics and thereby obtain high thruput operation. This intent ignores two very important problems: (1) the characteristics of many jobs are not known and (2) the characteristics of many jobs change dramatically during execution. These problems prompted the development of *HETM* to provide a dynamic dispatching capability while still retaining other good user oriented features of job class scheduling.

HETM dynamic dispatching

HETM periodically rearranges the preemptive *CPU* dispatching priority of specified tasks in an attempt to more equitably distribute *CPU* service and maintain system thruput. The set of tasks monitored by *HETM* (the dynamic priority group) is specified by the installation as is the monitoring period (the control interval). The equation employed to determine the *CPU* utilization history $(h_{i,l})$ of the ith task at control interval l is:

$$h_{i,l}=CPU_{i,l}+h_{i,l-1}-H_l/I \tag{1}$$

where:

$$H_l=\sum_{i=1}^{I}(CPU_{i,l}+h_{i,l-1})$$

I = number of tasks in dynamic priority group

$CPU_{i,l}$ = *CPU* usage of the ith task during the lth control interval.

Low h values indicate that the task has not utilized the *CPU* either because it was blocked by *I/O* or waiting, but not activated. High h values indicate that a task has had the opportunity and has utilized the *CPU*.

HETM is activated every control interval. The *CPU* utilization history values $(h_{i,l})$ of all tasks in the dynamic priority group are computed and the *OS* dispatching chain is reordered in inverse order of the value ranked history values.

Reference 7 establishes that the behavioral characteristics of the *HETM* algorithm for a system in statistical equilibrium may be described in terms of repeated patterns of priority mappings A probability of occurrence P_j may then be associated with the jth mapping. For a system in statistical equilibrium, the standard *HETM* algorithm will attempt to cause equal average *CPU* usage by each class as described by equation (2):

$$\sum_{j=1}^{M_p}P_jc_{ij}=\sum_{j=1}^{M_p}P_jc_{kj} \tag{2}$$

$$\forall i,k\in[1,M_c]$$

(It is possible that the different job class *CPU* usage characteristics will not permit satisfaction of equation (2). For example, in a two class system, the *CPU* usage of one job in highest priority could be lower than the *CPU* usage of the other job in lowest priority. Under such conditions *HETM* would always give the lower *CPU* usage job highest priority, but equation (2) would not be satisfied). It can be seen from equation (2), that *HETM* attempts to cause the control variable of equation (1), $CPU_{i,l}$, to have the same average value for all job classes.

Reference 7 establishes that *HETM* dynamic dispatching can have a marked effect on performance measures related to thruput and turnaround. Heuristically, it is certainly appealing in a rich *I/O* resource environment to give higher *CPU* priority to low *CPU* usage tasks so as to maintain *I/O* resource utilization without seriously degrading system *CPU* utilization. Reference 7 notes, however, that since the desired performance is not specified a priori for *HETM*, it is difficult to comment meaningfully on the real value of *HETM*.

It is also the case that *HETM* dispatching like that proposed by Wulf,[2] is based solely on the behavior of the current active jobs and is not influenced by the workload waiting to be processed.

Equal job class turnaround dispatching

Reference 5 points out that thruput should be defined not solely on the capabilities of a particular computer system, but rather on the ability of that system to process the given workload. For a given set of jobs, the average rate at which the system completes the jobs is an appropriate thruput measure. An appropriate definition of thruput for an equal worth, job class scheduled system is the time average rate of job completion requiring equal average turnaround for all job classes. For optimization purposes, the equal turnaround restriction can be posed as the constraint that jobs from different classes are processed at average rates proportional to the number of jobs waiting execution in the different class queues.

A maximum thruput problem constrained to equal average job class turnaround is posed and solved in Reference 5 and it is found that a full multiprogramming degree solution is optimal for an interesting range of relative class workload.

If P_j denotes the probability of the system operating in priority mapping j, the equal job class processing rate constraint is expressed in equation (3):

$$\sum_{j=1}^{M_p} P_j \frac{c_{ij}}{N_i C_i} = \sum_{j=1}^{M_p} P_j \frac{c_{kj}}{N_k C_k} \qquad (3)$$

$$\forall_i,\ k \in [1, M_c]$$

where: N_i is the number of jobs of class i to be completed
C_i is the average *CPU* time of a class i job
c_{ij} is the average *CPU* usage of a class i job when run in priority mapping j.

As indicated previously, the effect of the *HETM* dynamic dispatching algorithm is to provide constant time average values of its control variable over the classes, if this is possible. Equation (3) describes a situation where it is desired that the class proportional *CPU* usage have constant average value for all job classes. Therefore, if the class proportional *CPU* usage ($CPC_{i,l}$) defined in equation (4)

$$CPC_{i,l} = \frac{CPU_{i,l}}{N_{i,l} C_i} \qquad (4)$$

is employed as the control variable in the *HETM* algorithm of equation (1), equal average job class turnaround will be achieved where possible. Implementation of this algorithm could obtain the job class i queue lengths, N_i, from *HASP* and estimate the class i average *CPU* usage, C_i by a running average over completing jobs.

The assertion that the developed algorithm provides maximum thruput in addition to equal average job class turnaround must be qualified as follows:

(1) The developed algorithm requires multiprogramming of all available job classes (i.e., multiprogramming of degree M_c) and, as established in Reference 5, there are combinations of job class resource requirements where the optimal solution to the maximum thruput problem will not always multiprogram to the highest degree possible.
(2) Reference 5 only establishes the optimality of the developed algorithm (as qualified in (1) above) for the two job class case. Optimality may be inferred for a larger number of job classes, but it has not been formally established.

The developed algorithm is a good solution to the posed problem. The question naturally arises as to the generality of the problem. For a dynamic, development oriented computing environment such as found in an academic institution where good thruput could be strongly dependent on the number of small jobs done, the posed problem might not be appropriate; i.e., the control algorithm would delay processing a class of small jobs in favor of a class of many large jobs. For production data processing however, jobs are often classed to avoid resource conflicts and classes are considered to be equally worthy of system attention. The posed problem well describes such an environment.

Equal job class response ratio dynamic dispatching

The previous section indicates that while the requirement for equal average class turnaround deals with the question of processing the given workload, it may not be realistic. This would particularly be the case for large imbalances in the numbers of jobs in the class queues. This is the same sort of motivation that prompted the development of highest response ratio job initiation[4] and warrants the investigation of a possible extension to dynamic dispatching.

It is straightforward to define the concept of average job class response ratio as the ratio of the average class turnaround to the average class uniprogramming processing time. If γ_i is the average ratio of the *CPU* time, C_i, to the uniprogramming elapsed time of a class i job, the average uniprogramming elapsed time of a class i job is $C_i\gamma_i^{-1}$. To specify an equal average class response ratio constraint, the denominators of the terms in equation (3) need only be divided

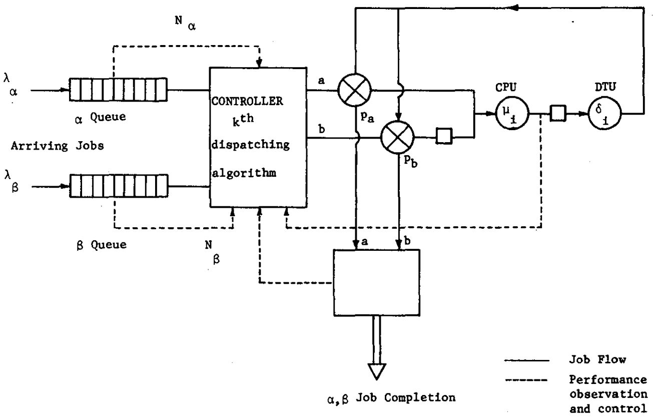


Figure 2—Two job class system with dynamic dispatching

by the appropriate class uniprogramming elapsed times. Therefore if the class proportional response ratio ($CPR_{i,l}$) defined in equation (5):

$$CPR_{i,l}=C_i\gamma_i^{-1}CPC_{i,l}=\frac{CPU_{i,l}}{\gamma_i N_{i,l}} \tag{5}$$

is employed as the control variable in the *HETM* algorithm of equation (1), equal average class response ratio will be achieved where possible.

The algorithm is direct to implement. Given an existing *HETM* implementation as defined in equation (1), all that remains is to substitute the $CPR_{i,l}$ of equation (5), for the $CPU_{i,l}$ in equation (1). To compute *CPR* it is necessary to obtain the job class queue lengths which are known to *HASP* and γ_i, the ratio of average class i job *CPU* time to uniprogramming elapsed time which can be given a priori. (This ratio should be reasonably constant for jobs classed on the basis of user type or resource usage.)

The presented class response ratio dynamic dispatching does not satisfy the previously stated objective of control based on performance improvement rather than a means to performance improvement. It does however provide an intuitively appealing strategy that is workload dependent. All other things equal, response ratio dynamic dispatching will promote the execution of classes of short jobs over classes of long jobs, but it will not allow a class of long jobs to grow indefinitely because of a large influx of short jobs.

The following section demonstrates the relative performance of the presented dispatching algorithms for a simple two class example.

TWO JOB CLASS EXAMPLE

A two job class example is presented and solved analytically, and the relative performance of the four dispatching algorithms is compared.

The dispatching algorithms of the preceding section were developed primarily for heavily loaded systems (i.e., systems with low probability of empty job class queues). The queueing theory approach employed in this example facilitates the investigation of relative performance over a wide range of system loading conditions.

Figure 2 presents a schematic of the general two class model. Jobs of type i (i is α or β) arrive from a Poisson source at average rate λ_i to the class i queue. Jobs of type i are characterized by:

λ_i Average arrival rate
μ_i Average service rate in preemptive exponential *CPU* (Central Processing Unit) service.
δ_i Average service rate in *FCFS* exponential *DTU* (Data Transfer Unit) service
p_i Probability of completion of job after *DTU* service. (Completed job is immediately replaced by another job if queue is non-empty.)

Internal to the system a higher preemptive *CPU* priority job denoted as job a and a lower priority job b compete for service from the *CPU* and the *DTU*. The *CPU* is modeled as an exponential server with service rate μ_i. Following completion of *CPU* service, a job either begins *DTU* service immediately or waits in the *DTU* queue for completion of the other job being processed by the *DTU*. The *DTU* is modeled as an exponential server with service rate δ_i. Following completion of *DTU* service, with probability p_i job i completes and leaves the system. It is immediately replaced by another type i job if there is one waiting for service.

This model is extremely simple. Recent work[8] has established, however, that the model predicts relative task resource utilization behavior with good fidelity. On this basis, it is argued that this simple model will indicate relative dispatching algorithm performance.

A controller (the kth dynamic dispatching algorithm) observes the behavior of the system through the *CPU* usage of the two jobs and the length of the two job queues. At each control interval Δ, the controller may change the relationship (mapping) between the external job types (α, β) and the internal priority designations (a and b). The standard *HETM* controller for example will reorder the mapping every Δ time units so the job with the lowest *CPU* utilization history is given the highest priority (i.e., corresponds to job a).

This section analyzes system performance for four different dispatching algorithms: (1) *OS* job class priority, (2) *HETM*, (3) equal average job class turnaround, and (4) equal average job class response ratio. The performance effects of these algorithms are discussed following development of the general model.

General model

The general system model is developed as two constituent models: (1) a model for multiprogramming of both α and β jobs, and (2) a model for uniprogramming of either type job when there are no jobs of the other type to process. In the sequel, these individual models are developed and a composite model is then synthesized.

Multiprogramming model

This model assumes that jobs of both types are present. A job in the system can be in one of four possible positions: (0) waiting for *CPU* service, (1) receiving *CPU* service, (2) waiting for *DTU* service, or (3) receiving *DTU* service. The contents (*a*, *b*, or *x*) of the last three positions are used to represent the state of the system (an *x* denotes the position being vacant). In this notation for example, state *bxa* is the state of job *b* being processed by the *CPU* and job *a* being processed by the *DTU*.

The states are related to each other by the possible state transitions caused by completion of service. For example if P_{bxa} denotes the steady state probability of being in state *bxa*, the rate of transition to state *xba* is $P_{bxa}\mu_b$ caused by *b* completing *CPU* service and the rate of transition to state *axx* is $P_{bxa}\delta_a$ caused by *a* completing *DTU* service and preempting *b* from *CPU* service.

The condition that the state transition rates be balanced for steady state plus the requirement that the sum of the state probabilities equal 1.0 leads to the following equations:

$$\begin{aligned}
&P_{axx}\mu_a = P_{axb}\delta_b + P_{bxa}\delta_a\\
&P_{bxa}(\mu_b+\delta_a) = P_{axx}\mu_a + P_{xab}\delta_b\\
&P_{xba}\delta_a = P_{bxa}\mu_b\\
&P_{axb}(\mu_a+\delta_b) = P_{xba}\delta_a \qquad (6)\\
&P_{xab}\delta_b = P_{axb}\mu_a\\
&P_{axx}+P_{bxa}+P_{xba}+P_{axb}+P_{xab} = 1.0
\end{aligned}$$

These equations are directly solved for the state probabilities.

The system can operate in two possible mappings as noted in Table 1.

The mapping probability P_j will be established by the long term effect of the controller (dispatching algorithm).

The job *CPU* utilization and effective service rate are useful measures of performance for each mapping *j*.

CPU Utilization:

$$\begin{aligned}
&\overline{CPU}_a{}^j = P_{axx}{}^j + P_{axb}{}^j\\
&\overline{CPU}_b{}^j = P_{bxa}{}^j \qquad (7)
\end{aligned}$$

Effective Service Rate:

The effective service rate $\bar{R}_{ei}{}^j$ of job *i* in mapping *j* is the rate at which it is processed by the *CPU* $(CPU_i{}^j\mu_i{}^j)$ times the probability of completing after *CPU* service $(p_i{}^j)$ or:

$$\begin{aligned}
&\bar{R}_{ea}{}^j = \overline{CPU}_a{}^j\mu_a{}^jp_a{}^j\\
&\bar{R}_{eb}{}^j = \overline{CPU}_b{}^j\mu_b{}^jp_b{}^j \qquad (8)
\end{aligned}$$

TABLE I—System Mappings

Mapping *j*	Priority Relationship	Mapping Probability
1	$a = \alpha,\ \ b = \beta$	P_1
2	$a = \beta,\ \ b = \alpha$	$P_2\ (= 1 - P_1)$

TABLE II—Two Examples

Example	μ_α	δ_α	p_α	μ_β	δ_β	p_β
1	10	20	.5	15	10	.5
2	10	20	.5	15	10	.3

When the system is operating under the *k*th dynamic dispatching algorithm, the algorithm will establish a steady state probability of operating in the *j*th mapping of $P_j{}^k$. Given the $P_j{}^k$, it is possible to develop system performance measures in terms of the external job types α and β for the *k*th algorithm as follows:

Job Multiprogramming *CPU* utilization:

$$\begin{aligned}
&\overline{CPU}_{m\alpha}{}^k = P_1{}^kCPU_a{}^1 + P_2{}^k\overline{CPU}_b{}^2\\
&\overline{CPU}_{m\beta}{}^k = P_1{}^k\overline{CPU}_b{}^1 + P_2{}^k\overline{CPU}_a{}^2 \qquad (9)
\end{aligned}$$

Job Multiprogramming Service Rates:

$$\begin{aligned}
&\bar{R}_{m\alpha}{}^k = P_1{}^kR_{ea}{}^1 + P_2{}^kR_{eb}{}^2\\
&R_{m\beta}{}^k = P_1{}^kR_{eb}{}^1 + P_2{}^kR_{ea}{}^2 \qquad (10)
\end{aligned}$$

Uniprogramming models

These models assume there is only one type of job, *i* (either α or β), to be processed. There is no waiting in the *CPU* or *DTU* queues and the job uniprogramming *CPU* utilization and uniprogramming job service rates are as follows:

Job Uniprogramming *CPU* Utilization:

$$\overline{CPU}_{ui} = \frac{\delta_i}{\mu_i+\delta_i} \quad \text{for } i=\alpha, \beta \qquad (11)$$

Job Uniprogramming Service Rates:

$$\bar{R}_{ui} = \frac{\mu_i\delta_ip_i}{\mu_i+\delta_i} \quad \text{for } i=\alpha, \beta \qquad (12)$$

Composite model

The composite model is to account for the effects of empty job queues. It is composed of combinations of results from

TABLE III—Uniprogramming Class Characteristics

			$\overline{\text{CPU}}_u$		$\overline{\text{CPU}}_c$		R_u	
Example	λ_α	λ_β	α	β	α	β	α	β
1	2.2	1.9	.667	.400	.440	.253	3.33	3.00
	2.0	1.0	.667	.400	.400	.133	3.33	3.00
	1.0	2.0	.667	.400	.200	.266	3.33	3.00
2	2.5	1.0	.667	.400	.500	.222	3.33	1.8
	2.1	1.2	.667	.400	.420	.266	3.33	1.8
	1.0	1.0	.667	.400	.200	.222	3.33	1.8

TABLE IV—Standard OS Fixed Priority Dispatching (k = 1)

EXAMPLE				$\bar{R}_c$		ρ_c		W_c		RR_c	
	λ_α	λ_β	P_1	α	β	α	β	α	β	α	β
1	2.2	1.9	1.0	2.85	1.65	.771	1.15	1.53	—	5.1	—
			0.0	2.03	2.80	1.08	.677	—	1.11	—	3.32
	2.0	1.0	1.0	3.11	1.87	.643	.533	.900	1.14	3.00	3.43
			0.0	2.66	2.86	.751	.349	1.51	.536	5.04	1.61
	1.0	2.0	1.0	2.99	2.41	.335	.828	.503	2.41	1.68	7.24
			0.0	2.01	2.91	.497	.687	.989	1.10	3.30	3.30
2	2.5	1.0	1.0	2.86	.883	.874	1.13	2.76	—	9.23	—
			0.0	2.18	1.68	1.14	.597	—	1.48	—	2.66
	2.1	1.2	1.0	2.84	1.03	.738	1.17	1.34	—	4.47	—
			0.0	1.96	1.68	1.07	.713	—	2.07	—	3.72
	1.0	1.0	1.0	3.05	1.45	.328	.687	.489	2.20	1.63	3.95
			0.0	2.23	1.75	.447	.571	.810	1.33	2.70	2.40

TABLE V—HETM Dynamic Dispatching (k = 2)

EXAMPLE				$\bar{R}_c$		ρ_c		W_c		RR_c	
	λ_α	λ_β	P	α	β	α	β	α	β	α	β
1	2.2	1.9	.184	2.08	2.50	1.06	.754	—	1.66	—	4.97
	2.0	1.0	.184	2.71	2.65	.737	.377	1.40	.605	4.68	1.81
	1.0	2.0	.184	2.15	2.78	.465	.719	.870	1.28	2.90	2.84
2	2.5	1.0	.184	2.22	1.48	1.12	.674	—	2.07	—	3.73
	2.1	1.2	.184	2.02	1.51	1.04	.796	—	3.25	—	5.86
	1.0	1.0	.184	2.35	1.68	.425	.595	.739	1.47	2.46	2.65

TABLE VI—Equal Job Class Turnaround (k = 3)

EXAMPLE				$\bar{R}_c$		ρ_c		W_c		RR_c	
	λ_α	λ_β	P_1	α	β	α	β	α	β	α	β
1	2.2	1.9	.556	2.30	1.99	.958	.954	10.3	10.9	34.4	32.8
	2.0	1.0	.783	2.97	2.05	.673	.488	1.03	.952	3.43	2.86
	1.0	2.0	.133	2.11	2.82	.474	.710	.902	1.23	3.06	3.68
2	2.5	1.0	.728	2.52	1.01	.994	.989	65.0	89.8	217.	162.
	2.1	1.2	.483	2.20	1.26	.957	.951	10.5	16.3	35.1	29.4
	1.0	1.0	.115	2.31	1.71	.433	.586	.765	1.42	2.55	2.55

TABLE VII—Equal Job Class Response Ratio (k = 4)

EXAMPLE				$\bar{R}_c$		ρ_c		W_c		RR_c	
	λ_α	λ_β	P_1	α	β	α	β	α	β	α	β
1	2.2	1.9	.557	2.30	1.99	.957	.955	10.2	11.2	33.9	33.5
	2.0	1.0	.803	2.98	2.03	.671	.492	1.02	.968	3.39	2.90
	1.0	2.0	.154	2.13	2.80	.471	.714	.889	1.25	2.96	3.74
2	2.5	1.0	.729	2.52	1.01	.993	.990	58.8	99.0	196	178
	2.1	1.2	.492	2.20	1.25	.953	.956	9.72	18.3	32.4	33.0
	1.0	1.0	.268	2.41	1.65	.415	.606	.708	1.54	2.36	2.77

the multiprogramming and uniprogramming models. The basic idea is that the external behavior of the cyclic server, two queue, service process can be represented as two composite single server, single queue; queueing processes. It is recognized that the density functions characterizing these equivalent service processes are more nearly hyperexponential than exponential, but for a first approximation it is assumed that the equivalent service processes are exponential with average service rates $\bar{R}_{c\alpha}{}^{k}$ and $\bar{R}_{c\beta}{}^{k}$ for the kth dispatching algorithm.

Queueing theory gives the utilizations of the two composite servers as:

$$\rho_{c\alpha}{}^{k} = \frac{\lambda_\alpha}{\bar{R}_{c\alpha}{}^{k}} \qquad \rho_{c\beta}{}^{k} = \frac{\lambda_\beta}{\bar{R}_{c\beta}{}^{k}} \tag{13}$$

and the average composite class turnaround as:

$$W_{c\alpha}{}^{k} = (\bar{R}_{c\alpha}{}^{k} - \lambda_\alpha)^{-1} W_{c\beta}{}^{k} = (\bar{R}_{c\beta}{}^{k} - \lambda_\beta)^{-1} \tag{14}$$

The composite server utilizations of equation (13) facilitate the computation of the composite service rates in terms of the multiprogramming model rates of equation (10) and the uniprogramming model rates of equation (12) as follows:

$$\bar{R}_{c\alpha}{}^{k} = \rho_{c\beta}{}^{k} \bar{R}_{m\alpha}{}^{k} + (1 - \rho_{c\beta}{}^{k}) \bar{R}_{u\alpha}$$

$$\bar{R}_{c\beta}{}^{k} = \rho_{c\alpha}{}^{k} \bar{R}_{m\beta}{}^{k} + (1 - \rho_{c\alpha}{}^{k}) \bar{R}_{u\beta} \tag{15}$$

In order to evaluate the effect of the equal job class response ratio dispatching algorithm it is useful to define composite class response ratios as in equation (16):

$$RR_{c\alpha}{}^{k} = W_{c\alpha}{}^{k} \bar{R}_{u\alpha}$$

$$RR_{c\beta}{}^{k} = W_{c\beta}{}^{k} \bar{R}_{u\beta} \tag{16}$$

The sequel develops the mapping probabilities $P_j{}^k$ for each of the four dispatching algorithms. The performance measures of equations (13-16) are tabulated for the two examples of Table II for several arrival rates.

Table III presents performance results for the two examples that are independent of the dispatching algorithm.

OS job class priority ($k=1$)

Under standard *OS* Job Class Priority there is no dynamic dispatching, *CPU* priority is directly determined by job type and the system operates in one mapping or the other. Thus if α has highest priority $P_1{}^1=1$, $P_2{}^1=0$ and if β has highest priority $P_1{}^1=0$, $P_2{}^1=1$. Table IV contains performance results for both these cases obtained by using the above mapping probabilities in evaluating equations (13-16).

HETM priority control ($k=2$)

As presented in equation (2), standard *HETM* forces the average *CPU* utilization of the two job classes to be equal (if possible) while the system is multiprogramming. By setting the $CPU_{m\alpha}{}^2$ and $CPU_{m\beta}{}^2$ of equation (9) equal, the mapping probabilities $P_1{}^2$ and $P_2{}^2$ can be easily determined. Use of these probabilities in equations (13-16) yields the performance results presented in Table V.

Equal job class turnaround ($k=3$)

As presented in equation (3), this dispatching algorithm forces, if possible, equal average job class turnaround while the system is multiprogramming. Equation (14) presents average composite class turnaround. This dispatching algorithm, however, forces equal average job class turnaround only during periods of multiprogramming. It is therefore necessary to determine $P_1{}^3$ and $P_2{}^3$ that cause equal average job class turnaround in the multiprogramming model only. The class multiprogramming turnaround is given in the equation (17).

$$W_{m\alpha}{}^{k} = (\bar{R}_{m\alpha}{}^{k} - \lambda_\alpha)^{-1} \qquad W_{m\beta}{}^{k} = (\bar{R}_{m\beta}{}^{k} - \lambda_\beta)^{-1} \tag{17}$$

The $P_1{}^3$ and $P_2{}^3$ are determined that cause $W_{m\alpha}{}^3$ and $W_{m\beta}{}^3$ to be equal. Use of these probabilities in equations (13-16) yield the performance results presented in Table VI.

Equal job class response ratio ($k=4$)

As indicated in equation (5), equal job class response ratio dispatching forces, if possible, the ratio of average job class turnaround to average class uniprogramming processing time to be equal across classes while the system is multiprogramming. For the cyclic server model, the uniprogramming processing time of a type i job is the reciprocal of the rate given in equation (12). Thus the mapping probabilities $P_1{}^4$ and $P_2{}^4$ can be determined by setting equal the products of class multiprogramming turnaround from equation (17) and uniprogramming class rate from equation (12) as in equation (18)

$$W_{m\alpha}{}^4 \bar{R}_{u\alpha} = W_{m\beta}{}^4 \bar{R}_{u\beta} \tag{18}$$

and solving for $P_1{}^4$ and $P_2{}^4$. Use of these probabilities in equations (13-16) yield the performance results presented in Table VII.

Comparison of performance

Comparison of relative performance of the four dispatching algorithms presented in Tables IV-VII offers few surprises. As might be expected for the similar job types of both examples, the *HETM* algorithm gives well balanced performance against both the composite class turnaround (W_c) and composite class response ratio (RR_c) indicators. The equal multiprogramming job class turnaround algorithm gives nearly equal composite class turnaround for high system utilization (i.e., as $\rho_c \rightarrow 1.0$), but for low system utilization it appears to do little better than *HETM*. The equal multiprogramming job class response ratio algorithm also gives more nearly equal composite class response ratios for high system utilization than for lower utilization. It should be noted that the ranges of job class behavior over which both equal multiprogramming class turnaround and equal multiprogramming class response ratio can be achieved is limited not only by the relative class characteristics (μ_i, δ_i, and ρ_i) as with *HETM*, but also by the relative class arrival rates (λ_i).

CONCLUSIONS

Three dynamic dispatching algorithms are presented and related to a simple representation of a job class scheduled multiprogramming system similar to OS/360. These three algorithms are applied to a simple two job class example and relative system performance is compared.

As expected, the standard *HETM* algorithm insures good system performance as compared to no dynamic dispatching with poor information on workload characteristics. The equal job class turnaround algorithm better balances response to actual workload for heavily loaded systems. The equal job class response ratio algorithm offers an interesting compromise between the *HETM* and equal turnaround algorithms while still allowing relative class response to be influenced by the relative class workload demands.

The analytic model results have been verified by recent descriptive simulation modeling of Chiang.[9]

REFERENCES

1. Bowdon, E. K. Sr., S. K. Mamrak, and R. R. Salz, "Performance Evaluation in Network Computer," *Proceedings of the Symposium on the Simulation of Computer System*, June 1973.
2. Wulf, W. A., "Performance Monitors for Multiprogrammed Systems," *Proceedings of the Second Symposium on Operating Systems Principles*, ACM, New York, pp. 175-181, 1969.
3. Northouse, R. A., and K. S. Fu, "Dynamic Scheduling of Large Digital Computer Systems Using Adaptive Control and Clustering Techniques," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-3, No. 3, pp. 225-234, May 1973.
4. Brinch Hansen, P., "An Analysis of Response Ratio Scheduling," *Proceedings of IFIP Congress 71*, Ljubljana, Yugoslavia, August 1971.
5. Chiang, A. T., and J. C. Strauss, "Priority Control For Maximum Thruput With Equal Job Class Turnaround," *Proceedings of Computer Science and Statistics Symp.*, Iowa State University, October 1973.
6. The *HASP* System Documentation for IBM Type 3 Program, *HASP* II, Version 3.0, No. 360-D-05.104., IBM Corporation, February 26, 1971.
7. Strauss, J. C., "An Analytic Model of the *HASP* Execution Task Monitor," *Communications of the ACM*, Vol. 17, No. 12, December 1974.
8. Wong, K., and J. C. Strauss, "Use of a Software Monitor in the Validation of an Analytic Computer System Model," *Software*, Practice and Experience, Vol. 4, No. 3, (1974).
9. Chiang, A. T., *A Simulation Study of Dynamic Dispatching*, Masters Thesis, Computer Science Dept., Washington University, St. Louis, Mo., May 1974.

JSYS Traps—A TENEX mechanism for encapsulation of user processes*

by ROBERT H. THOMAS
Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

INTRODUCTION

The JSYS Trap mechanism is an extension to the TENEX operating system[1,2] which enables a process** to define and control the virtual machine seen by other processes. Using the mechanism, a process can control the execution environment of other processes by providing them with a virtual machine that enlarges, restricts or completely redefines the "standard" virtual machine provided by TENEX.

The controlling process does this by declaring that it wishes to "monitor" (trap) selected system calls (JSYS's) when executed by other (inferior) processes. When a monitored process attempts to execute one of the calls specified it is suspended and the monitoring process is notified. After gaining control, the monitoring process may take whatever action it finds necessary. For example, it may choose to perform the call itself on behalf of the trapped process. Alternatively, it may allow the trapped process to perform the call itself, or it may first modify the call parameters and then allow the trapped process to resume normal execution of the system call.

Mechanisms similar to JSYS traps have been proposed in the context of TENEX and elsewhere.[3,4] The motivating forces that transformed the JSYS trap mechanism from an idea to a design and implementation for TENEX were the requirements placed on TENEX by the Resource Sharing Executive (RSEXEC) system.[5] The RSEXEC system is being developed as part of a research project in distributed computation.

One of the goals of RSEXEC is to enable the various TENEX Host computers† on the ARPA Computer Network[6,7] to function together as a single, multi-host TENEX system. RSEXEC provides an environment within which the resources available to a user are enlarged to include those beyond the boundaries of his local TENEX Host. It does this in a way that removes the logical distinction between resources which are "local" and those which are "remote". This applies to both the user at the "command language" level and his programs at the "executing process" level.

An important part of the RSEXEC environment is a distributed, multi-Host file system which allows files to be referenced without requiring Host specification. That is, a process need not be aware of the location within the network of files it uses in order to access them. Whenever a process attempts an operation involving a non-local file, the operation is dispatched across the network to a cooperating process running on the appropriate remote Host. As a result, existing "subsystems", such as text editors, assemblers and compilers, need not be rewritten to operate in the multi-Host environment.

We initially conceived of RSEXEC as an evolutionary system whose development would require considerable experimentation. Consequently, we decided that, at least initially, the RSEXEC environment would not be implemented as part of the normal TENEX operating system. Rather it would be provided by "ordinary user" processes which would act on behalf of processes attempting to access non-local resources. To provide the environment in this way it is necessary that ordinary user processes be able to intercept system calls made by other processes before the operating system itself acts upon them. JSYS traps were implemented to provide such an encapsulation mechanism.

Although the JSYS trap mechanism was strongly motivated by the RSEXEC application, it represents an important and powerful addition to the TENEX operating system which is useful in a general manner in applications requiring a controlled execution environment. This paper describes the trapping mechanism and records design and implementation decisions that were made in adding it to the existing TENEX operating system. In doing so, the paper describes some aspects of TENEX not previously reported.

The next section is a brief sketch of the TENEX virtual machine. Following that, the JSYS trap mechanism is described in more detail. First, it is described in terms of the properties we wanted it to exhibit and the constraints that consistency with the existing TENEX virtual machine placed upon it. Next, we describe the user's view of

* This work was supported by the Advanced Research Projects Agency of the Department of Defense under Contract No. DAHC15-71-C-0088.

** Almost any of the common definitions for the term process is adequate for the needs of this section. TENEX supports the concept of a tree structured process hierarchy described more fully in subsequent sections. With the exception of when it interacts with other processes, a TENEX process proceeds asynchronously as if executing on its own machine.

† There were thirteen TEXEX Hosts on the ARPANET as of October 1974.

JSYS traps. Finally, we view its implementation. The paper concludes by comparing the trapping mechanism with similar features in other operating systems.

THE TENEX VIRTUAL MACHINE

TENEX is a time-shared operating system developed by BBN to run on the DEC PDP-10 processor augmented with paging hardware. TENEX provides a multi-process job structure with software program interrupt capabilities, advanced file handling features and an interactive and carefully human-engineered command language. At present (October 1974) there are fourteen TENEX systems. This section focuses on the system call and multiprocess facilities of the TENEX virtual machine. Readers interested in other aspects of TENEX are referred to the literature,[1,2,8,9,10]

A user process running under TENEX executes on a virtual machine similar to a PDP-10 processor[11] with 256 K words of virtual memory. The direct input/output instructions of the PDP-10 are not available to user processes. Rather, the virtual machine provides input/output facilities which are considerably more powerful and sophisticated.

All of the virtual machine facilities* are accessed via a system call machine instruction, JSYS,** which was added to the PDP-10 processor for TENEX. The JSYS instruction accomplishes a transfer of control from a user process to the monitor routine that implements a particular system call in a single instruction time. The hardware interprets the address field of the JSYS instruction as an index into a transfer vector called the JSYS dispatch vector. The JSYS dispatch vector occupies exactly 1 page (512 words) in the monitor address space.*** TENEX users have come to regard the different system calls supported by the JSYS instruction as separate instructions. Thus, one speaks of the "Byte In" JSYS and the "Open File" JSYS, etc. This convention is used throughout the remainder of this paper.

When a user logs into TENEX a job consisting of a single process is created for him. By using appropriate system calls that process may create other processes which themselves may create further processes, etc. TENEX provides a separate virtual machine with its own address space for each such process. Each process has exactly one immediate superior (its creator) and may have any number of immediate inferiors (processes it has created). Thus the process hierarchy is tree-structured; the root of the tree being the process created at login time.

TENEX currently provides three mechanisms for interprocess communication:

1. Communication by direct process control whereby one process modifies the state of another.
   The state of a process includes its execution status (i.e., running, suspended by another process, blocked for input/output, etc.), program counter (PC), active registers (ACs) and the contents of its address space. A process can modify the PC and ACs of other processes and can start, stop and destroy them. The capability for direct process control is defined by the process hierarchy; processes may directly control only their inferiors.
2. Communication by pseudo-interrupt whereby one process transmits an interrupt signal to another.
   The signalling process specifies the target process and an interrupt channel. To receive the signal properly, the receiving process must have previously "armed" the specified interrupt channel by activating it, assigning it a priority, and specifying a routine to be executed whenever a signal for the channel occurs. The identity of the signalling process is not conveyed as part of the interrupt signal. In addition to other processes, a process may receive pseudo-interrupt signals from devices such as terminals and as a result of its own execution (e.g., arithmetic register overflow).
3. Communication through shared memory whereby communicating processes read and write from the same memory.
   The paging hardware partitions memory into pages of 512 words each. Each process sees a linearly addressable virtual memory of 512 pages which is defined by a memory map with an entry for each page. Each map entry describes a page in the process address space: an indication of whether the page exists, its physical location (i.e., current location in core or secondary storage) and the type of access the process has to the page. Processes can arrange to share portions of their address spaces by system calls that manipulate memory maps. For example, process A can share page 3 of its address space with page 5 of process B; any change to the shared page made by either process will be seen by both.

The multi-process features play an important role in standard TENEX operation. The process created for the user at login time runs the TENEX command language interpreter (EXEC). When a user invokes a subsystem (e.g., text editor) or a program of his own, the EXEC creates a process, whose initial virtual memory contains the program. After starting it, the EXEC blocks until the process terminates (see Figure 1). One interesting subsystem is IDDT, an interactive, "invisible" debugger,[12] which runs in a process inferior to the EXEC and superior to the process(es) being debugged. IDDT uses TENEX facilities for memory sharing and direct process control to enable a user to monitor (examine registers, address space,

* With the exception of the pager trapping facilities that implement the virtual memory and which are invisible to user processes.

** pronounced JAY-sys

*** For addresses greater than 511, the address is interpreted as an index into the user process address space and the process is dispatched to a routine in its own address space.

etc.) and control (start, stop, place "breakpoints", modify address space, etc.) the execution of a process in a manner that is transparent to the process.

DESIGN CONSIDERATIONS

Our goal was to add to TENEX a facility enabling one process to control the execution environment of another. The mechanism we chose was one in which the process being controlled is suspended whenever it attempts to execute JSYS's (system calls) previously specified by the controlling process to which control is then passed. The major constraint in designing the trapping mechanism was that it be done within the context of the existing operating system. Specifically, the mechanism had to be compatible with the TENEX virtual machine and its implementation could not require radical departure from the approach taken to the rest of TENEX. Of course, its implementation should not require excessive per process storage, should involve minimal overhead to processes not using it, and should not be excessively costly to those that do use it.

The following summarizes the major considerations that influenced definition of the JSYS trap mechanism:

1. The capability of a process for setting JSYS traps should be limited to processes inferior to it in order to provide a measure of protection that is consistent with the TENEX process hierarchy.
2. A process setting traps for another should be able to specify an arbitrary subset of JSYS's to be trapped rather than being required to specify only all or no JSYS's. This enables the inferior process to execute efficiently, incurring the overhead of being trapped only for those system calls the superior is interested in intercepting. In addition, it provides a measure of convenience for the trapping process; it need be programmed only to handle those JSYS's it is interested in. Furthermore, to allow for generality and flexibility a process should be able to dynamically remove traps it has set. It should not, of course, be able to remove traps set by other processes.

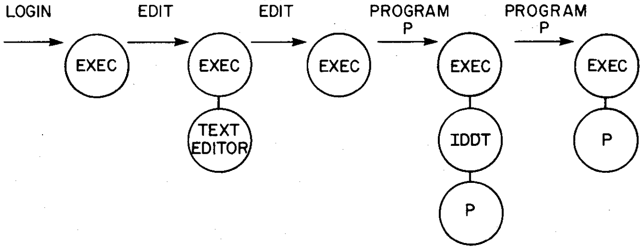


Figure 1—The process structure for a user job changes throughout the course of a TENEX session. The TENEX EXEC (command language interpreter), which resides in the top process in the job process hierarchy, creates and manages other processes as the user's requests dictate

3. The traps set for a process should be inherited by its inferiors. That is, when a process is created, it should be subject to the same traps as its creator. Additionally, when traps are set for a process, they should also be set for all existing processes inferior to it. This ability to set traps *indirectly* allows a process to control the virtual machine seen by all its inferiors without requiring that it know the details of the inferiors' process structure. In addition, it prevents a trapped process from using inferior processes to execute (trapped) system calls on its behalf in order to bypass the trapping mechanism.
4. A trapping process should be able to allow a process that has been suspended as a result of executing a trapped JSYS to resume "normal" execution of the JSYS that caused the trap. This is useful in situations in which one process is monitoring another. For example, a process which may not be completely trustworthy could be encapsulated by a monitoring process which would trap operations that are potential security violations in order to prevent it from writing "private" data to a "public" or nonsecure "area". The monitoring process would allow the inferior to resume execution of such an operation only after checking the call parameters to ascertain that the operation is "safe".
5. Control should propagate up the process hierarchy from controlling process to controlling process. When a process attempting to execute a particular JSYS is suspended, control should be passed to the nearest superior in the hierarchy that requested to trap that JSYS. If that process resumes the suspended process without changing its PC and execution status, control should pass to the next superior in the hierarchy handling that JSYS. Should each controlling process in the hierarchy resume the trapped process without resetting its PC, "normal" execution of the JSYS should be resumed. This insures that each process in the hierarchy wishing to trap the JSYS has a chance to handle it. Additionally, it prevents a process from bypassing the trapping mechanism by creating inferiors and then trapping and immediately resuming their calls executed on its behalf.
6. The trapping mechanism should be transparent to the trapped process. In particular, the execution of a given JSYS should appear to be the "same" to the executing process whether or not the JSYS is trapped by a superior process. This permits existing programs to run in a trapping environment without requiring that they be rewritten.
7. To allow for flexibility and generality, a process should be able to use JSYS traps to control its various inferior processes differently. That is, it should be able to specify a different set of JSYS's to be trapped for each inferior.

The transfer of control from the trapped process to the trapping process involves suspension of the former and no-

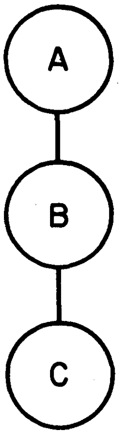


Figure 2—Process A may not directly set traps for Process C. However, process C inherits any traps set by process A for process B

tification of the latter. We chose to have notification of the trapping process occur via a pseudo-interrupt signal. Two other approaches suggested themselves:

1. The trapping process could use a periodic polling procedure to look for processes suspended as the result of traps it had set; or
2. The trapping process could execute a system call causing it to block until the "next" trap occurred.

The first alternative was rejected immediately on efficiency grounds because it requires a "busy wait" by the trapping process. The second was judged to be less flexible than the pseudo-interrupt approach because it requires the trapped process to relinquish control and therefore to remain idle while awaiting the next trap. If this effect is desired, a user can achieve it in a straightforward way using existing system calls in conjunction with the trap pseudo-interrupt. Furthermore, since the implementation would be virtually identical for both the pseudo-interrupt and blocking approaches, we selected the more flexible pseudo-interrupt approach.

As an implementation consideration, we restricted the ability of a process to set traps beyond that suggested in consideration (1) above. A process can *directly* set and remove traps only for processes that are *immediately* inferior to it. To allow a process to set traps for non-immediate inferiors would not violate the TENEX process hierarchy. However, it would require a considerably more complex implementation, particularly in terms of maintaining the data base required to describe the trapping situation (see "Implementation" Section below). Because traps are inherited by inferiors in the process hierarchy (consideration (3) above) this is not a severe restriction. For example, for the situation in Figure 2, process A may directly set traps for process B but not for process C; however, any traps set for B are inherited by C. This restriction also prevents a process from trapping its own execution of JSYS's. While there are situations in which this would be useful, we felt that the additional implementation complexity required to support the capability was unjustified.

USERS VIEW OF THE TRAPPING MECHANISM

The trapping mechanism was made available to user processes by augmenting the virtual machine with several new JSYS's (system calls). The basic calls are summarized below in an informal notation that conveys their meaning while avoiding the details of TENEX programming. Values returned by a call are indicated on the left side of an "=" sign.

To set or remove JSYS traps the following calls are used.

set-traps (proc, trap-spec)
remove-traps (proc, trap-spec)

Proc is the process ID of an immediately inferior process and *trap-spec* is the address of a table specifying the JSYS's for which traps are to be set or removed. A process can declare the channel on which it wishes to receive pseudo-interrupt signals resulting from JSYS traps by the call:

set-trap-channel (chn)

where *chn* is a channel number for the interrupt channel.

The call to determine the source of a trap pseudo-interrupt is:

proc, call=trap-data ()

Proc is the ID of the process that was suspended attempting to execute the JSYS *call.* Before such a pseudo-interrupt can occur, the trapping process must have previously set a trap for the *call* JSYS in *proc* and declared a channel for trap interrupts. To respond to a trap, the trapping process may use any of the operations normally available for direct process control: it can read the parameters supplied by the trapped process, set the value (if any) to be returned to the trapped process, change its PC, modify its address space, change its execution status, etc. After the trap has been handled, the trapped process may be allowed to resume execution using the call:

resume-trapped-proc (proc)

where *proc* is the process to be resumed.

A process may use the call

t=test-trap ()

where the value returned is either true or false, to determine whether traps have been set for it by a superior process. Since all JSYS's including those for managing traps may be trapped by superiors, this call need not violate the transparency property (6 above) desired for the trapping mechanism: A process could prevent inferiors from determining whether they are being trapped by trapping their execution of test-trap and always returning the value false.

An example should clarify how the trapping mechanism can be used. Consider the simple task of generating a frequency histogram of system calls made by an arbitrary program. A process Q can do this by creating another process P to run the program and then trapping and recording the JSYS's P executes. The following annotated program fragment describes Q:

```
   .
   .
   .
 (enable pseudo-interrupt-system)
 set-trap-channel(n)               //Assign n as channel for trap
                                   //pseudo-interrupts.
 P=create-proc(file)               //Create P initializing its address
                                   //space to the program stored as
                                   //file
 set-traps(P,ALL)                  //Set traps for ALL JSYS's
                                   //executed by P.
 start-proc(P)                     //Start program execution by P.
 wait-proc(P)                      //Wait until P terminates.
 output(Histogram)                 //Output the Histogram array.
   .
   .
   .
PSI-Handler-n:                     //JSYS trap interrupt handling
                                   //routine
 R,i=trap-data()                   //Read trap data.
 Histogram(i)=Histogram(i)+1       //Account for JSYS in Histogram
                                   //array.
 resume-trapped-proc(R)            //Allow P to resume normal
                                   //execution of JSYS i.
 break                             //Break from the trap interrupt.
```

Use of the trapping mechanism in this example is relatively simple: process *Q* merely resumes *P* after recording the trap. In the distributed file system application described earlier, the process running RSEXEC uses trapping to extend the TENEX virtual machine to support access to files remote from the local TENEX Host. It traps file operations made by processes inferior to it (e.g., text editors, compilers, etc.). Whenever a file operation is initiated that requires access to a remote file, RSEXEC sends a request across the network to a cooperating "service" process at the proper Host instructing it to execute the operation on behalf of the inferior process (See Figure 3). Operations that can be handled locally are passed directly to the local operating system by RSEXEC. After the operation has been performed RSEXEC resumes the inferior process, by properly incrementing its PC and providing return parameters (if any). Because the trapping activity is transparent, the inferior process can uniformly access all files, both local and remote, without regard for their location within the network.

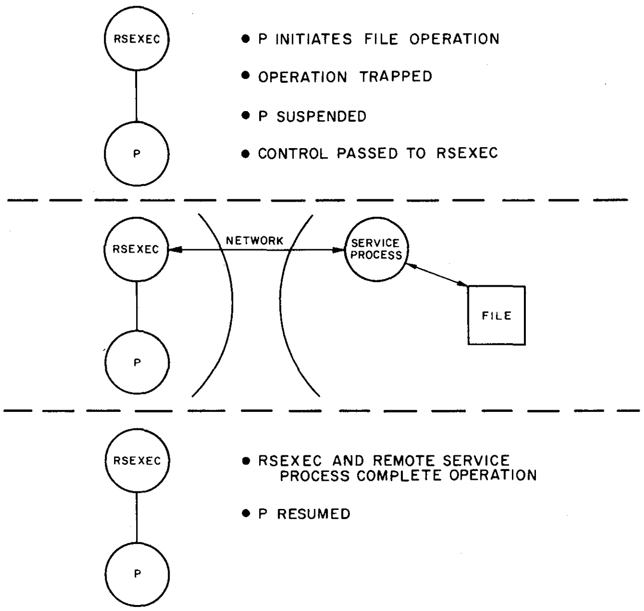

RSEXEC
P
• P INITIATES FILE OPERATION
• OPERATION TRAPPED
• P SUSPENDED
• CONTROL PASSED TO RSEXEC
RSEXEC
NETWORK
SERVICE PROCESS
FILE
P
RSEXEC
• RSEXEC AND REMOTE SERVICE PROCESS COMPLETE OPERATION
• P RESUMED
P

Figure 3—RSEXEC uses the JSYS trap mechanism to support uniform access by a user program (process P) to local and remote files. Access to remote files is accomplished by interacting with a remote service process

Other applications for the trapping mechanism readily suggest themselves. JSYS traps have proven to be a powerful debugging aid. For example, a complex program, which was believed to have been debugged and which is run continuously on TENEX as a service "demon" process, began to malfunction by closing a critical data file for no apparent reason on the order of once a day. After unsuccessfully studying program listings and using conventional debugging techniques for several days, the programmer built a simple process to trap and examine all operations that could possibly result in closing the file. He then ran the malfunctioning service process as an inferior to the trapping process and was able to intercept the operation that caused the malfunction the first time it occurred (approximately ten hours after the program was placed in execution). We plan to add this debugging technique to the repertoire of IDDT, the invisible debugger, such that a user can cause a program being debugged to "break" on certain system calls. This technique would enable the user to gain control on, for example, all file output operations without requiring that he remember and specify the program location of each. When the program breaks he could inspect the parameters, perhaps request IDDT to execute the call and inspect the result, and then allow his program to proceed to the next breakpoint.

A somewhat different use of the trap mechanism would enable a user to use programs written by others with the assurance that doing so would not compromise the security of his data. For example, he could encapsulate such programs in a controlled environment which selectively inhibits output operations by trapping them and allowing only those directed to "legitimate" destinations to continue. He could even intercept and prevent use of more subtle techniques for leaking information such as those recently noted by Lampson.[13]

IMPLEMENTATION

The implementation of the JSYS trap mechanism is described in this section with the focus on approach rather than detail. The result is a simplified sketch of the implementation. First, it is necessary to present as background

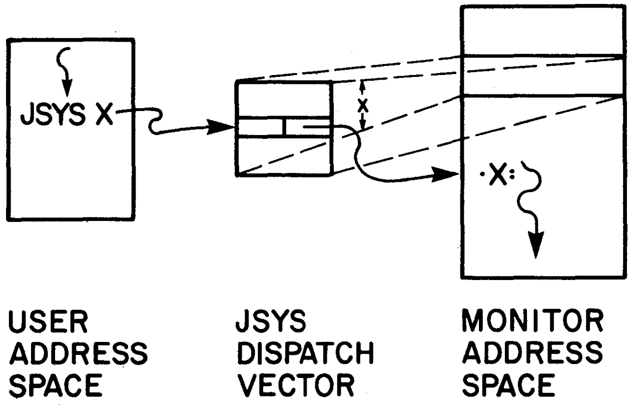


Figure 4—The JSYS instruction uses the JSYS dispatch vector to accomplish a transfer from user to monitor address space

some facts about the JSYS instruction and the structure of TENEX.

A TENEX process has two address spaces: A user address space where the process executes the instructions of the user program; and, a monitor address space that is invisible to the user program where the process executes monitor routines in response to system calls initiated by execution in the user address space. Transfer from user to monitor address space is accomplished by execution of a JSYS instruction with an effective address of less than 512 (see Figure 4). The processor enters "monitor mode" and uses the effective address as an index into the JSYS dispatch vector to fetch two pointers; it stores the user process PC and processor flags through one of the pointers and resumes execution in monitor address space at the location specified by the other.

Like the user address space, the monitor space for a process is a paged, 256K word linearly addressable space. Unlike the user space, the monitor space is partitioned into several areas (see Figure 5). These areas are:

1. a process private area that holds process state information (PC, ACS, user address space map, etc.);
2. a job private area, shared by all processes in a job, used for job wide data (the structure of the job process hierarchy, data about files open by processes in the job, etc.);
3. a "public" area, shared by every process in the system, that contains monitor routines and various system wide data bases.

The public area is further subdivided into in core resident and swappable regions. Both the process and job private areas are swappable.

The hardware paging device is aware of the organization of the monitor address space and, depending upon the area being referenced, takes different actions to complete a memory reference. References to the resident area are generally direct and bypass the page mapping operation although the pager can be instructed to "map the resident monitor" (see below); references to the public swappable area are mapped via a resident monitor page map; references to the job and process private areas are mapped via a page map for the process private area.

Implementation of the trapping mechanism specified in the previous section was feasible for TENEX because only a finite number of system calls (512) are possible* and all system calls "pass through" a single point in the system: the JSYS dispatch vector.

To implement JSYS traps, the dispatch vector, formerly a page shared by all processes in the system, was made process private. Trapped processes have a modified dispatch vector. Entries corresponding to trapped JSYS's point to a "trap and interrupt" routine and those for untrapped JSYS's point to the standard monitor routines for those calls. We saw two ways to make the dispatch vector private:

1. Make a minor (hardware) modification to the JSYS instruction so that it uses as its dispatch vector, a page in the process private area rather than one in the public area.
2. Leave the JSYS instruction unmodified. When a trapped process is running, set up the monitor map entry for the JSYS dispatch vector to point to a page in the per process area (the modified dispatch vector) and cause the pager to map references to the resident monitor (see above).

The primary advantage of the second approach is that it allows the trapping software to run on the existing hardware at all TENEX installations. Its disadvantages are two: slightly slower execution for trapped processes resulting from mapping each reference to the resident monitor; and severe constraints on the software for (planned) dual processor configurations resulting from limitations of both the paging device and PDP-10 processor. The current implementation provides for both alternatives, allowing each installation to choose one at "system generation time".

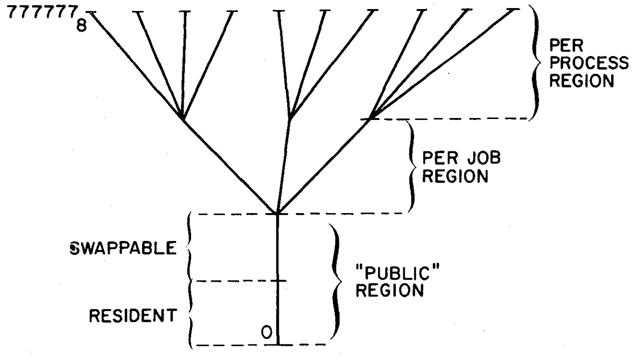


Figure 5—Schematic of the TENEX monitor address space

* The TENEX operating system, of course, can (and does) support more than 512 system operations. This is accomplished by multiplexing similar operations on a given JSYS by using call parameters to indicate the desired operation.

The "trap and interrupt" routine executed when a JSYS is trapped, suspends the trapped process and interrupts the proper process in the hierarchy. Should the suspended process be resumed without modification to its PC, the routine continues by searching the hierarchy for additional processes to interrupt. The situation in Figure 6 illustrates that the *immediate monitor* of a trapped process (i.e., the nearest superior process trapping its JSYS's) is not necessarily the correct process to interrupt. Consider the proper sequence of events for execution of JSYS's 1,2,3 and 4 by process E, assuming that all processes respond to the trap interrupt merely by resuming E:

JSYS 1: trap to A
(note bypass of immediate monitor D and intermediate monitor C)
JSYS 2: trap to D, then trap to C, then trap to A
(all monitors in hierarchy receive interrupt)
JSYS 3: trap to C, then trap to A
(note bypass of immediate monitor D)
JSYS 4: trap to D, then trap to A
(note bypass of intermediate monitor C)

The point here is that the immediate monitor may not have set the trap for the particular JSYS executed. The modified dispatch vector (JDVEC) is sufficient to initiate trap action but is insufficient, by itself, to specify the proper process(es) to be notified. To enable the "trap and interrupt" routine to complete the trap action properly, additional information is associated with each process:

1. The name of the process which is its immediate monitor (IM);
2. A list of the JSYS's for which its immediate monitor has set traps (TTBL).

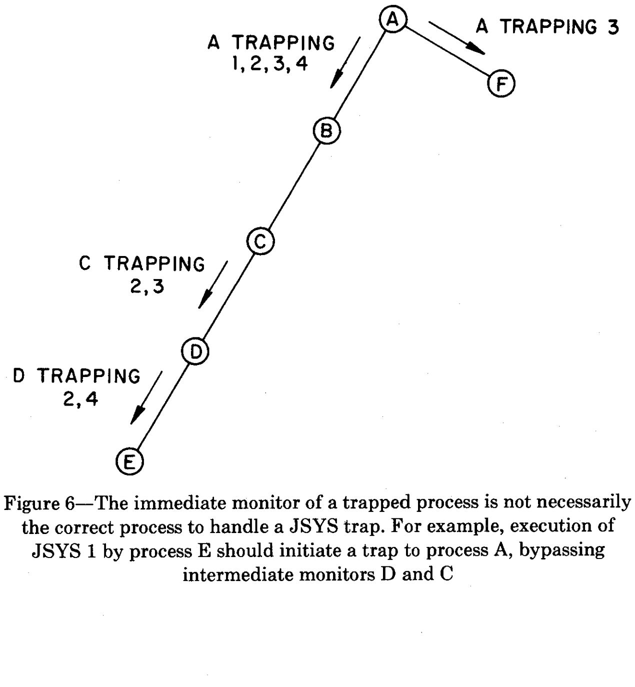


Figure 6—The immediate monitor of a trapped process is not necessarily the correct process to handle a JSYS trap. For example, execution of JSYS 1 by process E should initiate a trap to process A, bypassing intermediate monitors D and C

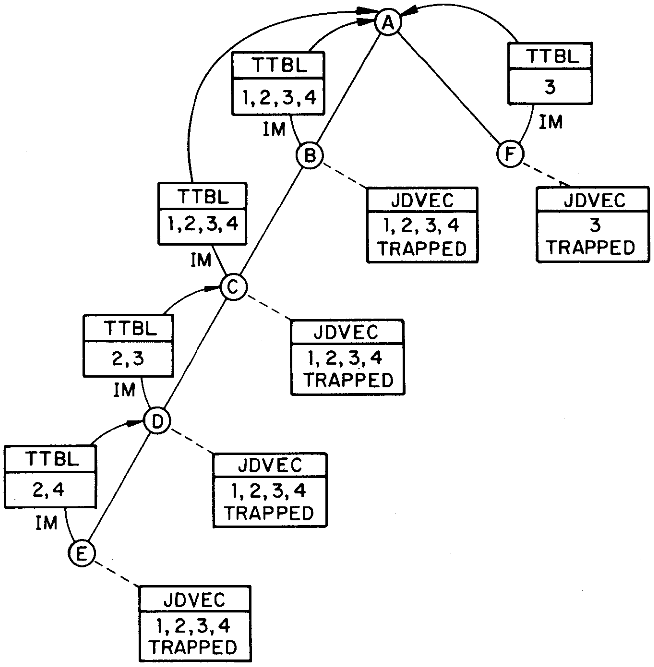


Figure 7—When a process attempts to execute a trapped JSYS, a "trap and interrupt" routine makes use of the modified JSYS dispatch vector (JDVEC), the name of the immediate monitor of the process (IM) and a list of JSYS's being trapped by the process (TTBL) to determine which process to interrupt

The situation in Figure 6 is redrawn in Figure 7 to illustrate how JDVEC, IM and TTBL are used when a process executes a trapped JSYS. Assume process E initiates JSYS 3 and, as before, assume that all processes receiving trap interrupts respond by resuming E. The following sequence of events occurs (refer to Figure 7):

1. E is dispatched through its JDVEC to the "trap and interrupt" routine.
2. IM of E is process D; D is not trapping E's execution of JSYS 3 (from E's TTBL); bypass D.
3. IM of D is C; C is trapping JSYS 3; interrupt C and suspend E;
4. C resumes E;
5. IM of C is A; A is trapping JSYS 3; interrupt A and suspend E;
6. A resumes E;
7. IM of A is null; dispatch E to standard monitor routine for JSYS.

Setting and removing traps for a process is accomplished by a recursive routine that "walks" over the process hierarchy appropriately updating JDVEC, IM and TTBL for the process and its inferiors. Reconsider the situation of Figure 7; the effect of

set-traps (C, JSYS-2-and-5)

executed by process B is shown in Figure 8. Removing traps is the trickier of the operations, as care must be

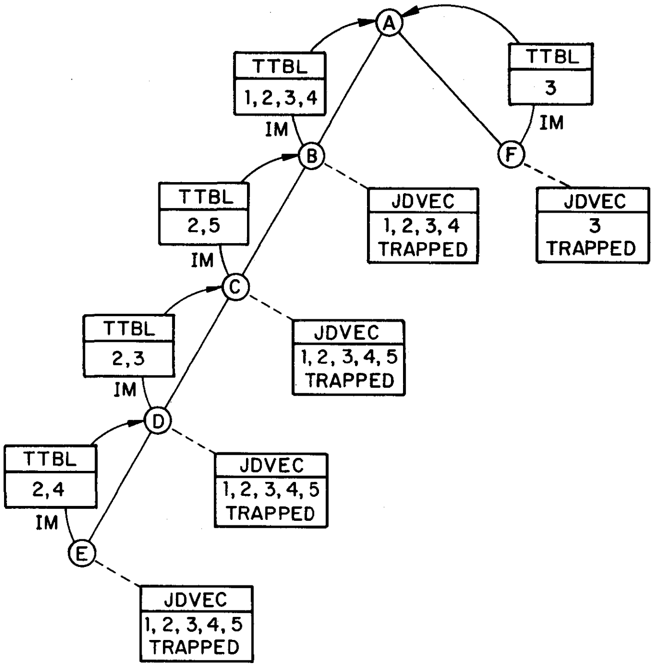


Figure 8—Note the changes in IM, JDVEC, and TTBL for processes C, D, and E resulting from process B setting traps for JSYS's 2 and 5 for process C

taken to restore to JDVEC the normal dispatches for only those JSYS's no longer trapped by any process in the hierarchy. The following cases (refer to Figure 7) illustrate the nature of the problem:

1. remove-traps (B, JSYS-1) executed by A:
   Restore the normal dispatch for JSYS 1 to JDVEC of B,C,D,E;
   Remove JSYS 1 from TTBL of B and C
2. remove-traps (B, JSYS-2) executed by A:
   Restore the normal dispatch for JSYS 2 to JDVEC for B and C;
   C and D trap JSYS 2, hence do not modify JDVEC of D and E;
   Remove JSYS 2 from TTBL of B and C
3. remove-traps (E, JSYS-2) executed by D:
   E's execution of JSYS 2 is trapped by A and C, hence do not modify JDVEC C of E;
   Remove JSYS 2 from TTBL of E.

Moving the JSYS dispatch vector from a resident page shared by all processes to a swappable page in the process private region adds to system memory management overhead by:

a. increasing paging activity—modified dispatch vectors must be swapped into core as the corresponding trapped processes execute; and
b. placing increased demand for storage on the swapping medium to hold the numerous dispatch vectors.

To reduce this overhead, the implementation minimizes the number of dispatch vectors the system must maintain by sharing them among processes whenever possible.* All untrapped processes share the same modified dispatch vector which is a page in the resident region. Furthermore, the routine that sets and removes traps is careful to insure that a process and all of its inferiors having the same immediate monitor share the same dispatch vector. Figure 9 shows how dispatch vector sharing relations change as traps are set and removed.

As we have described the implementation, a process whose execution of certain JSYS's is trapped can execute untrapped JSYS's without incurring overhead due to the trapping mechanism. Because the overhead resulting from execution of a trapped JSYS strongly depends upon the situation,** it is impossible to give a single, simple measure of how expensive trapping is. However, a comparison of the times required for a process to execute a given (nontrapped) JSYS and to execute the same JSYS for a well-defined, "best case" trapping situation represents a useful measure of the trapping overhead.

The time required to execute JSYS X is:

2 μsec (=time to accomplish transfer from user to monitor space; usually insignificant)
+CPU time to execute system routines that implement call X

The time required when JSYS X is trapped by only a

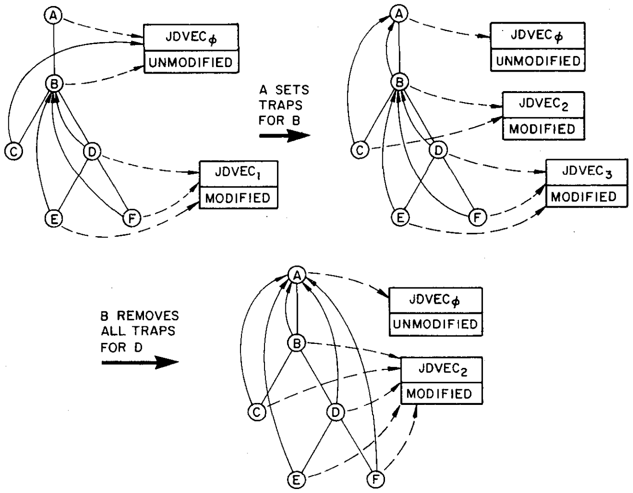


Figure 9—To minimize the number of JSYS dispatch vectors that must be maintained, TENEX insures that processes share dispatch vectors whenever possible

* Two processes can be made to share a page by setting the corresponding map entries in their memory maps to point to the same physical page of memory.

** i.e., the number of superior processes trapping the JSYS, the actions they take in response to the trap interrupt, and "unrelated" factors such as how heavily loaded the system is when the trap occurs.

single process that eventually returns control to the trapped process is:

2 μsec (insignificant here)
+CPU time to execute system routines that implement call X
+1.7 msec (=CPU time required to pass control from trapped to trapping process and back; determined by counting instructions making "best case" assumptions)
+CPU time trapping process uses in response to trap
+2 process wakeups*

It is clear that the (percentage) overhead incurred by trapping JSYS X is a strong function of the complexity of JSYS X. Measurements made for the "Byte IN" JSYS (a moderately "quick" call that reads the "next" byte from an open file) for the situation in which the trapping process immediately resumes the trapped process shows the untrapped operation to be 3 to 10 times faster than the trapped one.** By using highly tuned, tightly coded routines rather than the existing, "general purpose" monitor routines to transfer control among the processes, we estimate that the 1.7 msec figure could be halved. This would result in halving the overhead in trapping the "Byte IN" operation. Our experience with the trapping mechanism has shown that the delay resulting from the two process wakeups is the most significant component of the overhead. This component is largely due to the current TENEX scheduling algorithms which treat the two processes as independent, whereas, in reality, they operate in a tightly coupled, coroutine-like fashion. We feel that a significant reduction in trapping overhead would result by modifying the TENEX scheduler to support coroutine-like transfer of control between processes whereby one process could relinquish the processor (i.e., its remaining CPU quantum) and its memory resources (used to hold its working set) to another process without invoking the "normal" processor and memory management operations.

DISCUSSION

The features we are familiar with in other systems that most closely approximate the JSYS trap mechanism are "dynamic linking" in MULTICS[14,15] and "facility calls" in the operating system being developed by Project SUE.[16,17] The intended uses of these features are somewhat different from those of JSYS traps and of each other; thus the capabilities they provide, while similar in some respects, exhibit significant differences.

In MULTICS all "system calls" are made by invoking subroutines. The "linkage" to an "external" subroutine, such as one implementing a MULTICS "system function" is not established until the routine is called for the first time during program execution. At the first call a "fault" occurs that activates a dynamic linking procedure. As a result, various "system" and "user" file directories are searched for the routine. When (if) the routine is found, the linkage is made such that subsequent calls of the routine do not cause a "fault" and then normal execution of the subroutine call is resumed. Dynamic linking is motivated largely by the desire to support and encourage modular programming.

The dynamic linking mechanism of MULTICS can be used to substitute "non-standard" routines for the "standard system" routines by placing such routines in the file directories that are searched and (or) by specifying alternative directory "search paths". Use of dynamic linking in this way could approximate some capabilities JSYS traps offer. However, there are significant differences that should be noted. First, the "set-up" procedure is different and, in practice, would probably deter use of linking in this way. The set up involves storing a file for each call to be intercepted in an appropriate "user" file directory. The MULTICS system provides many ways of accomplishing a given function. As a result, in order to provide an execution environment for a given program it is necessary to know which of the many possible routines the program uses for each of the functions to be controlled. This, together with the potential for file naming conflicts, suggests that implementation of software to provide a controlled environment within which arbitrary users may run programs of their choice would be decidedly non-trivial.

Secondly, the linking activity takes place within the context of a single process. The multiple process nature of the trapping mechanism makes hierarchies of execution environments relatively easy to implement. Consider, as an example, the situation of a debugging environment (such as IDDT provides) existing within a multi-host file system environment (such as RSEXEC provides); system calls would be interpreted first by the process implementing the debugging environment and then by the process implementing the file system environment. It is difficult to see how dynamic linking could be used to implement such a hierarchy of execution environments. Finally, the linking mechanism is designed to serve as a subroutine linkage mechanism. The kind of "controlling" actions a "substitute" routine can take are constrained to be consistent with the subroutine discipline. Furthermore, once a link is established it generally exists for the duration of the computation. In this sense, JSYS traps are more "dynamic" in that they may be set and removed repeatedly.

Project SUE at the University of Toronto is developing an operating system in which system resources and services are provided by dedicated "system" processes. The "facility call" concept was developed to meet the interprocess communication requirements presented by such a system organization. To request a service a process

* Time for TENEX to "notice" that the trapping process has been interrupted, should be awakened and given the CPU and, later, that the trapped process has been resumed and should be awakened.

** The large variance is due to the fact that it is sometimes necessary to read a file page from secondary storage into the monitor file buffer to complete the read operation.

issues a facility call. As a result it is blocked until the process responsible for the service completes the request. Thus, the virtual machine seen by a process is defined by the processes which respond to the facility calls it makes. Because facility calls can be used by all processes for interprocess communication the distinction between "user" and "system" processes is a weak one. That is, a so called "user" process could provide service for another user process in much the same way a TENEX process can use JSYS traps to provide services for another process that are not directly supported by the operating system. Unlike JSYS traps, facility calls (as described in References 16 and 17) cannot be used to redefine existing system calls. Therefore the approach one would take to provide a controlled execution environment would be somewhat different than that using JSYS traps. It would involve reprogramming the processes that provide the functions to be controlled. The extent to which this is feasible would depend upon where the functions of interest are provided: at a low level by "standard" system processes, or at a high level by user processes. As noted earlier, a requirement of the JSYS trap mechanism was that it enable all standard system functions to be intercepted without modifying the way the operating system itself provides them. Because there is no obvious analogy in facility calls to the way a trap for a particular JSYS can be passed from process to process, hierarchies of controlled environments would be difficult to implement.

CONCLUDING REMARKS

Our experience with using the trapping mechanism has, to date, been somewhat limited. However, we feel that it represents an extremely powerful operating system facility. Although we have discussed trapping in the context of its realization in the TENEX operating system, we feel that the trapping concept is a general one which is consistent with a variety of operating system philosophies and should appear in some form in every "general purpose" operating system. We recommend that system designers seriously consider providing similar, user accessible mechanisms for program encapsulation in future operating systems.

ACKNOWLEDGMENTS

The author wishes to thank a number of colleagues for their contributions to the work presented above, especially R. S. Tomlinson and J. D. Burchfiel who constructively commented on the JSYS trap design and on the implementation approach; D. C. Allen and R. S. Tomlinson whose knowledge of the TENEX monitor was invaluable during the debugging phase; and R. E. Schantz, T. A. Standish, and W. R. Sutherland who constructively commented on the presentation of this paper.

REFERENCES

1. Bobrow, D. G., J. D. Burchfiel, D. L. Murphy and R. S. Tomlinson, "TENEX, a Paged Time-Sharing System for the PDP-10," *Communications of the ACM* 15, 3, pp. 135–143, March 1972.
2. Murphy, D. L., "Storage Organization and Management in TENEX," *AFIPS Conference Proceedings,* Vol. 41, 1972 AFIPS Press, Montvale, New Jersey, pp. 23-32.
3. Thomas, R. H., *A Model for Process Representation and Synthesis,* Ph.D. Thesis, Department of Electrical Engineering, M.I.T., June 1971. (Also available as Project MAC Technical Report TR-87.)
4. Bernstein, A. J. and P. Siegel, *Hardware for Level Structure Operating Systems,* Technical Report 21, State University of New York at Stony Brook, Department of Computer Science, October 1973.
5. Thomas, R. H., "A Resource Sharing Executive for the ARPANET," *AFIPS Conference Proceedings,* Vol. 42, 1973, AFIPS Press, Montvale, New Jersey, pp. 155-163.
6. Roberts, L. G. and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *AFIPS Conference Proceedings,* Vol. 36, 1970, pp. 543-549.
7. Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther and D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," AFIPS Conference Proceedings, Vol. 36, 1970.
8. TENEX JSYS Manual—A Manual of TENEX Monitor Calls, BBN—Computer Science Division, BBN, Cambridge, Massachusetts, September 1973.
9. Myer, T. H., J. R. Barnaby and W. W. Plummer, *TENEX Executive Language Manual for Users,* BBN Computer Science Division, BBN, Cambridge, Massachusetts, April 1973.
10. *TENEX User's Guide,* BBN Computer Science Division, BBN, Cambridge, Massachusetts, January 1973.
11. Digital Equipment Corporation, PDP-10 Reference Handbook, December 1971.
12. Plummer, W. W., IDDT User Manual, BBN Computer Science Division, BBN, Cambridge, Massachusetts, 1973.
13. Lampson, B. W., "A Note on the Confinement Problem," *Communications of the ACM,* 16, 10, October 1973, pp. 613-615.
14. Organick, E. I., *The MULTICS System: An Examination of its Structure,* M.I.T. Press, 1972.
15. Vyssotsky, V. A., F. J. Corbato and R. M. Graham, "Structure of the MULTICS Supervisor," *AFIPS Conference Proceedings,* Vol. 27, 1965.
16. Sevcik, J. W., J. W. Atwood, M. S. Grushcow, R. C. Holt, J. J. Horning, and D. Tsichritzis, "Project SUE as a Learning Experience," *AFIPS Conference Proceedings,* Vol. 41, 1972, pp. 331-338.
17. Holt, R. C. and M. S. Gruschow, "A Short Discussion of Interprocess Communication in the SUE/360/370 Operating System," Proceedings ACM SIGPLAN/SIGOPS Interface Meeting, April 1973.

Operating system penetration

by RICHARD R. LINDE
System Development Corporation
Santa Monica, California

INTRODUCTION

One of the favorite diversions of university students involves "beating" the system. In the case of operating systems, this has been a remarkably easy accomplishment. An extensive lore of operating system penetration, ranging from anecdotes describing students who have outsmarted the teacher's grading program to students who captured the system's password list and posted it on one of the bulletin boards,[1] has been collected on college campuses. Private industry has been victimized much more seriously. Here the lore of "system" penetrations contains scenarios involving the loss of tens of thousands of dollars.[2]

The Research and Development organization at SDC has been seriously involved with legitimate operating system penetration efforts. Under contract to government agencies and industry, SDC has assessed the secure-worthiness of their systems by attempts to gain illegal access to their operating systems. As of this date, seven operating systems have been studied. This paper examines the successful penetration methodology employed, and the generic operating system functional weaknesses that have been found. Recommendations are made for improvement that can strengthen the penetration methodology.

THE SDC FLAW HYPOTHESIS METHODOLOGY

In the absence of more formal correctness proof techniques,[3] penetrations are the most cost effective method for assessing vulnerabilities. Exhaustive testing of an operating system's security controls is different than subjecting these controls to a penetration attack. System testing exercises are used to examine a system for implementation errors; whereas, penetration tests are used to examine an implementation, and from these analyses infer areas of possible design weakness.

Peterson and Turn have defined comprehensive system attack strategies, and have proposed a set of countermeasures.[4] This paper focuses more on software attack strategies with respect to generic operating system weaknesses than on countermeasures. In this case, countermeasures entail good system design practices. SDC has formalized a strategy, the Flaw Hypothesis Methodology, based on our experience with operating system penetrations.[5]

Comprehensive flaw finding requires four stages: knowledge of the system control structure; the generation of an inventory of suspected flaws; i.e., "flaw hypotheses"; confirmation of the hypotheses; and making generalizations regarding the underlying system weakness for which the flaw represents a specific instance.

Knowledge of system control structure

Knowledge of the system control structure is an obvious prerequisite to the penetration effort. It is necessary for the penetration analysts to understand how users interact with the system, what services are provided to them, and what constraints are placed on them. In order to gain this familiarity, penetration analysts must read the system manuals pertaining to command language, system debugging, editing, operator's instructions, terminal user's instructions, and the introductory manuals pertaining to system overview and generation. Hence, the penetration analysts are able to list the security objects—i.e., file data, password lists, disk volumes—that are protected by system control objects—i.e., the file cataloger, installation management techniques, and label checking of disk volumes, respectively. Sometimes an object may be both a security object to be protected and a control object that protects other objects. For example, a password list may be viewed as a control object when it is protecting access to files, or it may be viewed as a security object protected recursively by itself, by installation management techniques guaranteeing password integrity, by password changeability, etc.

Security weaknesses found in operating systems contribute to the formulation of a control object hierarchy for a hypothetically "secure" system. This hierarchy of control objects can be represented in graphical form; graphical templates have been produced for each functional area of the hypothetical system (see Figure 1; the arrows between each node reflect a logical dependency). As more "live" systems are studied, each template is changed to reflect more "adequate" controls for the hypothetical system. The templates are applied to the "live" system under analysis by using them to produce graphs for the system. Flaws are postulated based upon control objects that may or may not be present. "Templates" or sequences of faulty generic code derived from past system studies are used as search parameters when perusing system listings. Cross reference programs for searching a data base of system symbols would be an excellent analytic tool at this phase of the methodology.[6] Conceptually, the hierarchical dependency graph is a representation standing between a protection matrix model[7] and a flow chart of the operating system. In order to understand an operating system well enough to comprehensively penetrate it, one must view the operating system from an abstract level as well as from an implementation level, e.g., the following levels:

- Inter-Module Design: Schematics of operating system modules and layers.[8,9]
- Access Control Mechanism: For example, set theoretic descriptions of the access matrix.[10]
- Control Object Hierarchy: Hierarchical dependency graphs (see Figure 1).
- Intra-Module Design: System flow charts and Logic manuals.
- Implementation: Symbolic listings of system code.

A penetration consists of an interloper capturing a control object, sometimes starting with a low-valued object on the hierarchical dependency graph and working his way up to complete omniscience—the supervisor bit (i.e., supervisor state). Graphs with badly designed control objects, such as faulty I/O control design decisions, are classified as generic functional weaknesses when they can be penetrated at many nodes.

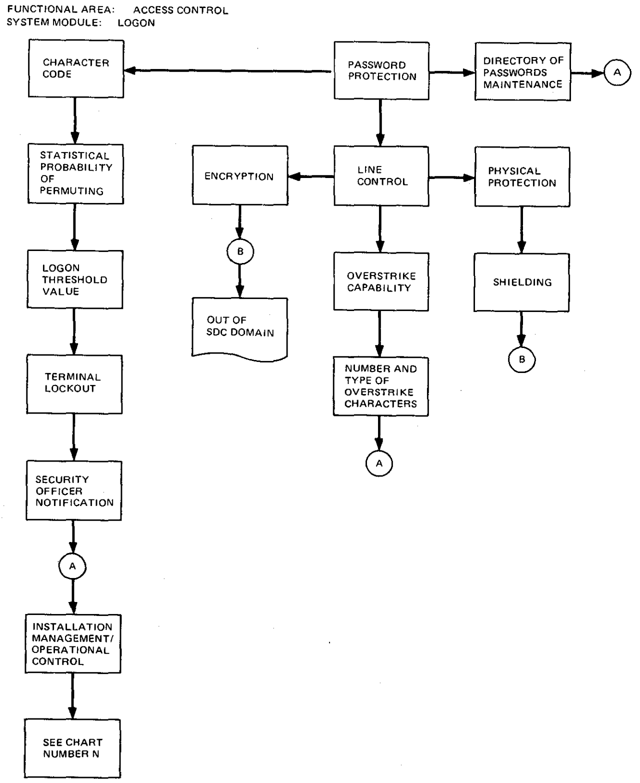

FUNCTIONAL AREA: ACCESS CONTROL
SYSTEM MODULE: LOGON
CHARACTER CODE
PASSWORD PROTECTION
DIRECTORY OF PASSWORDS MAINTENANCE
A
STATISTICAL PROBABILITY OF PERMUTING
ENCRYPTION
LINE CONTROL
PHYSICAL PROTECTION
B
LOGON THRESHOLD VALUE
OVERSTRIKE CAPABILITY
SHIELDING
OUT OF SDC DOMAIN
B
TERMINAL LOCKOUT
NUMBER AND TYPE OF OVERSTRIKE CHARACTERS
A
SECURITY OFFICER NOTIFICATION
A
INSTALLATION MANAGEMENT/ OPERATIONAL CONTROL
SEE CHART NUMBER N

Figure 1—Security Control Object Dependency Graph: A Generic Template to be Applied to the System Under Investigation

TABLE I—Flaw Hypotheses Generators

FLAW HYPOTHESES GENERATORS
HISTORICAL GENERIC SYSTEM WEAKNESSES (SEE APPENDIX A)
SYSTEM PROHIBITIONS AND WARNINGS Timing Dependencies
INTERFACES Man-Man (Operator Messages) Man-System (Commands)
SELDOM USED OR UNUSUAL FUNCTIONS OR COMMANDS Read Backward
CONTROL OBJECT DEPENDENCY GRAPH TEMPLATES
HISTORICAL ATTACK STRATEGIES (See Appendix B)
SYSTEM LISTINGS, LOGIC MANUALS, USERS' GUIDES
COLLECTION OF USER AND SYSTEM PROGRAMMER EXPERIENCES WITH THE SYSTEM UNDER ANALYSIS

Flaw hypothesis generation

The security control object dependency graph is a useful heuristic in that it gives the penetration analysts a visual representation of potential operating system attack points. Many of the flaw hypotheses that are generated result from such graphs. Once a hypothesis is formulated, it is written onto a form that contains a cross reference to the security control object dependency, a priority of investigation based upon the nature of the object and the probability of exploiting it, an attack strategy, and an evaluation summary. System code is also studied and used to generate flaw hypotheses. Specific areas of the operating system are analyzed from different points of view, and kinds of possible security weaknesses are generated for each area. The result is the generation of the flaw hypothesis form. Table I contains a list of flaw hypotheses generators.

Flaw hypothesis confirmation

"Gedanken" (thought) experiments and desk-checking are employed, using existing documentation, program logic manuals, and symbolic listings of the system, to determine the validity of a flaw hypothesis. The Gedanken experiment is the most important phase of the penetration study. This is attested to by the fact that most uncovered flaws are found by "thought" testing.

The intent of live tests is to ascertain that a flaw exists and a penetration is possible. A penetration program involves a minimum amount of thought once the flaw is proved. It does require large, straightforward programs to perform input/output, timing, initialization, set-up, etc. (programs written to prove flaws because thought testing has proven inconclusive, can be quite complex, however). Sometimes programs are produced to perform a permutation of all possible operation codes in an attempt to uncover instructions not contained in the System's Principles of Operation. This is an example of where a penetration tool, i.e., a program, can be produced to look for potential flaws, testing for flaws in the absence of the appropriate documentation.

Flaw generalization

It is important for a penetration team to analyze an operating system's security strengths and weaknesses thoroughly, and to delineate the generic weaknesses of the system. In order to describe these weaknesses, the penetration analysts devise a categorization based on families of errors for the study. When one functional area begins to yield a few penetrations, the investigators meet to discuss the nature of the penetration attempts, and to explore other areas of the system, e.g., access control in lieu of I/O control.

GENERIC OPERATING SYSTEM FUNCTIONAL WEAKNESSES: WHICH MODULES TO STUDY FIRST?

Security Control design and implementation weaknesses occurring repeatedly in computer systems can be grouped into a common class based on functionality. This may be a software, hardware, or physical function that is being provided. If it is a software function, the protection mechanism associated with it is usually distributed through more than one layer of system control, and the function itself is generally divided into multiple modules. Often each module is produced by different groups of people. For example, the generic function, physical resource sharing, may be distributed among the Scheduler, Catalogers, Dispatcher, Input/Output Supervisor, Terminal Read/Write Controller, etc. Each module may be part of a different control layer, implementing both security policy and protection mechanism decisions in the system. Systems designed in such a manner are vulnerable to a number of penetration attacks since security design decisions related to a common function have been distributed among several system building teams.

The following describes a set of common operating system functional weaknesses based upon the seven systems that have been studied:

I/O control

Most operating systems allow input/output channel programming in some form. This is a prime target area for potential system penetrators. The complexities of I/O code alone can be a significant factor. Logical errors, inconsistencies, and omissions can often be found by detailed examination of the I/O code. Even where checked it may still be possible for a channel program to modify itself, loop back, and then execute the newly modified code. Also, I/O channels usually act as independent processors and because of this may have unlimited access to memory. The dangers here are that critical code in the system could be modified in an unauthorized manner. Additionally, if an infinite loop can be executed, service to the other users can be intentionally degraded or stopped. This, of course, can be a very dangerous security threat. If a system can be selectively overloaded at critical (real) time periods, the loss of command and control could be severe. This would especially be true in tactical systems, air traffic control systems, or a Navy sea plot environment.

Program and data sharing

Typically operating systems attempt to isolate a user's process and its data from other users' processes and their data. This involves the protection of a number of security objects: hardware resources, files, password lists, operating system routines, etc. However, since dedicated machines or dedicated classified periods of processing are not cost effective to provide computing resources and software services in a completely isolable manner to a community of users, some form of multiprogramming with controlled sharing must take place. These sharing mechanisms involve program and data sharing as well as physical resource sharing, and because of faulty operating system design and implementation techniques, they are prime penetration targets. Incongruously, the operating system must isolate its users from one another yet provide them with controlled communication paths.

Access control

In order to permit sharing of resources, data, and programs, the users must be identified and authenticated by the operating system and the operating system must be identified by the user. To permit this "handshake protocol," a Send/Receive relationship is established between the user and the operating system. Failure to handle this interface communication function correctly provides the opportunity for a user process to masquerade as an operating system process, and for a user to obtain password information through "piece-wise decomposition" and "permutation" type programs (see Appendix B). Less sophisticated but effective penetration techniques involve permuting easy-to-guess passwords, operator spoofs, browsing for poor password overstrike capability, etc.

Installation management/operational control

In a temporal sense, operating system controls are only effective during the period of system execution and must be augmented by static procedural controls such as:

directory maintenance
password assignment
system generation parameters
system changes
building of software utilities
system checkout
enforcing system prohibitions
period processing requirements
disk and tape degaussing, labeling, inventory

Because control of this function is apart from the operating system in a temporal and physical sense, it is particularly vulnerable to interdiction and corruption of system code by planting trap doors and Trojan Horse routines. Personnel inefficiency and inadequate procedural controls account for this susceptibility.

Auditing and surveillance

There is a need to insure that system security controls are working properly. An audit mechanism can be built to record all security transactions, such as file OPEN requests, LOGON requests, resource expenditures, and other security related events. Surveillance can take place dynamically within the operating system through the monitoring of well-designed thresholds. Also, "friendly" programs can be built to periodically assault the system's security controls to insure that they are working properly.[10,11,12] Unfortunately, many systems lack a comprehensive audit and surveillance capability, employing an accounting mechanism instead.

Externally, a security officer introduces a human decision-making capability into the surveillance mechanism. The security officer's function can include the ability to log-off suspicious users and to monitor the security safeguards to insure they are functioning as intended.[11,12] Penetrator entrapment (also called counter-intelligence) can be used with respect to the security officer. Bogus listings with "magic passwords" inside can be used to trap a penetrator using those passwords.

In part, the value of audit and surveillance mechanisms must be weighed against the skill of the penetrator. Clearly, the more skilled penetrator will disable these mechanisms first in order to work undetected. These mechanisms work best against the casual attack by the less skilled penetrator who confronts the system protection matrix attempting to acquire "booty."

As system protection mechanisms are improved, the "penetration work factor"—the amount of effort and resources expended to gain unauthorized access to data, procedure, or machine resources—could become so great that other methods, such as buying off an employee with access to the information will be less expensive.[13] Audit and surveillance mechanisms increase the "penetration work factor" for the skilled penetrator. Penetration studies may be used to quantify this work factor.

Non-software weaknesses

The security aspects of hardware and physical constraints on the computer system have received less attention in our analysis of operating systems than have the software and administrative protection mechanisms. A list of hardware and physical threats have been compiled and have been reported by other studies.[14,15,16]

GENERIC SYSTEM FLAWS

Operating system functional areas are vulnerable to attack strategies because they contain specific flaws. Flaws that repeatedly occur in Computer Systems may be grouped into common classes. These flaws are described more fully in Appendix A. Because of these weaknesses, the capture of design control objects (i.e., a password list) is possible. Conceptually, generic flaws may exist at the nodes of the generic functional dependency graph templates (see Figure 1).

GENERIC ATTACKS

A generic attack is a hostile action that is found to have been repeatedly successful in penetrating operating systems. Typically a security control object (i.e., the storage protection bit) is attacked and captured because an operating system design or implementation mechanism is flawed in a specific functional area. The primary attack may lead to secondary attacks aimed at higher security control objects until a security object (i.e., Top Secret File) is captured.

ATTACK SCENARIOS

In order to illustrate the relationship between generic functional flaws and their proneness to an Appendix B penetration attack, a few representative attack scenarios are presented. One or more of the generic system flaws described in Appendix A provide a focal point for each attack:

Input/output control

- In a so-called chained command, the data channel interprets the address and count fields but discards the operation code and performs the operation defined in the predecessor command. On the hardware, any bit pattern can be formed in the ignored field. Place a branch in such a program attempting to reuse the formerly chained command in an unchained mode.[17]

Program and data sharing

- Transfer a copy of one's own EDIT to the private files belonging to User A. If private files are searched before public files, pseudo-EDIT will be loaded for User A.[18]
- Take advantage of the system's failure to validate user-supplied addresses to where a store or fetch is to be executed, compromising the system's and user's address spaces. For example, the location for storing return parameters from a service call may not be validated.
- A borrowed service routine sends data from the address space of the borrower to the address space of the service routine's owner, using the system's interprocess communication facilities.
- If the system has interlocks that prevent files from being opened for reading and writing at the same time, the service can leak data if it is allowed to read files which can be written by its owner. The interlocks allow a file to simulate a shared Boolean variable which one program can set and the other can test.[19] Or eight files can be used simultaneously for ASCII encoding of short character messages.
- Place a program in a loop that continually requests spooling or terminal read/write buffers.
- Although memory is cleared prior to use, the system fails to clear the state vector general registers until a program has been loaded. Execute a dump routine prior to loading a program and browse for residue.

Access control

- Have one's LOGON routine masquerading as the system's at a hard wired terminal. Simulate a system crash after capturing the password.
- Divert operator's attention with a MOUNT message. Imbed multiple line feeds in a message buffer followed by the system prefix character with a simulated system message comprising the text.

Installation management/operational control

- Although User's Guides, etc., warn against operating self-modifying channel programs, the installation fails to restrict their use. Write a self-modifying channel program and penetrate address boundaries.

Audit and surveillance

- One "secure" system places too much reliance on audit and surveillance not realizing it will be the first mechanism disabled by the interloper. Disable the mechanism and proceed in undetected fashion with another penetration attack.

Physical/hardware

- Tap a minicomputer into a communications line to masquerade as a legitimate central system. This is a method used to acquire user ID's and passwords.

SUMMARY AND OBSERVATIONS

SDC's operating system penetration methodology consists of:

(1) Knowledge of System Control Structure
(2) Flaw Hypothesis Generation
(3) Flaw Hypothesis Confirmation
(4) Flaw Generalization

This methodology is useful in providing a formal strategy for penetrating an operating system, as well as for isolating generic system functional flaws that can be used for determining functional areas of operating system design that need strengthening.* Seven systems were penetrated during the study and the results of these penetrations were used to strengthen this methodology and to isolate specific areas of operating system weakness.

One of the last operating systems that was studied had a high exposure rate (65 percent) with respect to flaw hypotheses generated, indicating the effectiveness of the penetration method and team. I/O control and Installation Management/Operational Control were the system functional targets for the most penetrations; "Unexpected Parameters" and "Denial of Access Programming" were the most frequently used penetration attack strategies.

The penetration methodology will be strengthened by collecting, analyzing, and classifying more contemporary system functional design weaknesses, by developing analytic tools for locating security control objects in system listings,[6] and by producing more permutation type programs. The next class of tools that need to be developed are programs that search for security flaws in the symbolic listings of the operating system.[6,22] The construction of such programs will be guided by collecting more generic code templates or signatures common to the generic functional weaknesses of contemporary operating systems.

ACKNOWLEDGMENT

The author wishes to acknowledge the valuable editorial assistance provided by Clark Weissman, who originated the idea for the Flaw Hypothesis Methodology, and to Dr. R. Stockton Gaines for his most helpful critique. It is next to impossible to provide proper attribution for a specific

* A contemporary solution to the design of secure operating systems exists with respect to secure kernels and virtual machine monitor architecture studies.[17,18,20,21]

attack; however, I would like to acknowledge the fact that Richard Bisbee, Dan Edwards, and several of the Multics people, whom I have not referenced herein, should be given credit for devising a number of the penetration attacks. To all others whom I have not referenced or acknowledged, please forgive me.

REFERENCES

1. Organick, Elliott I., *The MULTICS System: An Examination of Its Structure*, the MIT Press, Cambridge, Massachusetts, 1972.
2. Palme, Jacob, "Software Security," *Datamation*, Vol. 20, No. 1, January 1974, pp. 51-55.
3. Linden, T. A., "A Summary of Progress Toward Proving Program Correctness," *AFIPS Conference Proceedings*, 1972 Fall Joint Computer Conference, Vol. 41, pp. 201-211.
4. Peterson, H. E., and R. Turn, "System Implications of Information Privacy," *Spring Joint Computer Conference Proceedings*, 1967, AFIPS Press, pp. 291-300.
5. Weissman, C., *System Security Analysis/Certification Methodology and Results*, SDC SP-3728, 8 October 1973.
6. Webb, Douglas A., *Analytic Tools for the Examination of Security Aspects of Operating Systems*, Lawrence Livermore Laboratory, UCRL-76016, September 1974.
7. Lampson, B. W., "Protection," *Proceedings Fifth Annual Princeton Conference on Information Sciences and Systems*, Department of Electrical Engineering. Princeton University, Princeton, New Jersey, March 1971, pp. 437-443.
8. Dijkstra, E. W., "The Structure of THE—Multiprogramming System," *Comm. ACM*, II, 5, May 1968, pp. 341-346.
9. Parnas, D. L., "On the Criteria to be Used in Decomposing Systems into Modules," *Comm. ACM*, Vol. 15, No. 12, December 1972, pp. 1053-1058.
10. Weissman, C., "Security Controls in the ADEPT-50 Time-Sharing System," *Fall Joint Computer Conference Proceedings*, 1969, pp. 119-134.
11. Linde, R. R., C. Weissman, and C. E. Fox, "The ADEPT-50 Time-Sharing System," *Fall Joint Computer Conference Proceedings*, 1969, pp. 39-50, AFIPS Press.
12. Linde, R. R., "Operational Management of Time-Sharing Systems, *Proceedings 1966 National ACM Conference*, pp. 149-159.
13. Lackey, R. D., "Penetration of Computer Systems–An Overview," *Honeywell Computer Journal*, pp. 81-85, September 1974, Vol. 8, No. 2.
14. Molho, Lee M., "Hardware Aspects of Secure Computing," *Spring Joint Computer Conference Proceedings*, 1970, AFIPS Press.
15. Van Tassel, Dennis, *Computer Security Management*, Prentice-Hall, Inc., 1972.
16. Hollingworth, D., *Enhancing Computer System Security*, P-5064, The Rand Corporation, August 1973.
17. Belady, L. A., and C. Weissman, *Experiments with Secure Resource Sharing for Virtual Machines*, SDC SP-3769, 15 May 1974.
18. Popek, G. J., and C. S. Kline, "Verifiable Secure Operating System Software," *AFIPS Conference Proceedings*, 1974, National Computer Conference, Vol. 43, pp. 145-151.
19. Lampson, Butler W., "A Note on the Confinement Problem," *Comm. ACM*, Vol. 16, No. 10, October 1973, pp. 613-615.
20. Weissman, C., *Secure Computer Operation with Virtual Machine Partitioning*, SDC SP-3790, 6 September 1974.
21. Attanasio, C. R., "Virtual Machines and Data Security," *Proceedings ACM SIGOPS/SIGARCM Workshop on Virtual Computer Systems*, Cambridge, Massachusetts, 1973.
22. Hollingworth, D., S. Glaseman, M. Hopwood, "*Security Test and Evaluation Tools: An Approach to Operating System Security Analysis*, P-5298, The Rand Corporation, September 1974.
23. Brinch Hansen, Per, "The Nucleus of a Multiprogramming System," *Comm. ACM*, 13, 4, April 1970, pp. 238-241.
24. Branstad, Dennis K., "Privacy and Protection in Operating Systems," *IEEE Computer*, January 1973, pp. 43-46.

APPENDIX A—GENERIC SYSTEM FUNCTIONAL FLAWS

- Authentication—It is important for the user to be able to authenticate that the operating system and hardware he is executing on is what they purport to be. Little attention has been paid to the software identification of software modules. This becomes more important in a shared segmented system because of the ease with which a module can be substituted or replaced.

  Systemwide schemes are lacking, whereby major hardware components (i.e., storage controllers, processors, peripheral controllers, communication controllers, and remote terminals), can be identified by each other.
- Documentation—Security documentation may be written in a complex manner or be deficient in several areas.
- Encryption—Communication lines must be encrypted if shielding is inadequate. Passwords stored in memory are not encrypted in many systems.
- Error Detection—Protection mechanisms may be disabled or modified as a result of an error, and after subsequent return to the routine causing the error, they may not be reset.
- Implementation—All the well thought out design requirements may be reversed by a bad implementation. Condition codes may be tested improperly, etc. More importantly, implementation in many systems means that more definitive design decisions are to be made by the implementor.
- Implicit Trust—Routine B assumes routine A's parameters are correct because Routine A is a system process.
- Implied Sharing—The system stores its data or references user parameters in the user's address space because memory is in critical demand.
- Interprocess Communication—To facilitate sharing, users are permitted to request hardware and software resources from the operating system using a SEND/RECEIVE type mechanism.[23] Various return conditions (password OK, segment error, illegal parameter, etc.) may give the penetrator "significant" information, especially when this design weakness is combined with others such as Implied Sharing.
- Legality Checking—User parameters may not be checked adequately; addresses may overlap each other or refer to system areas; condition code checks may be omitted; and unusual or extraordinary parameters may not be anticipated by the designer or implementor.
- Line Disconnect—The operating system hardware must make the system cognizant of line disconnects prompting an automatic logout or a RELOG request (i.e., RELOG with correct password).
- Modularity—Many systems were designed so that different groups of people were responsible for the design of various modules embodying common functions at the same system level (e.g., channel command

translation and execution; spooling simulation). Thus, when design changes were necessary, they were made by different, isolated system designers.[9]

- Operator Carelessness—Operators may be tricked into mounting bogus operating system packs for the penetrator. Many systems do not adequately perform label checks and provide correct warning and error messages to the operator.
- Parameter Passing by Reference vs. Passing by Value—It's much safer to pass parameters in general registers than to use the registers to point to the parameter's location. Passing by Reference can lead to an Implied Sharing design weakness through careless implementation since the parameters may not be moved out of the user address space before legality checking occurs.
- Passwords—Passwords for operating system, terminal, and file access that are supplied by users may be easy to guess. Passwords that are made up of a limited set of characters or syntax rules (e.g., digits 0 to 9 only or A to Z only) are subject to permutation attempts.
- Penetrator Entrapment—This design practice sometimes called Counter Intelligence is useful for the capture of the unskilled penetrator. Generally, pseudo-system weaknesses are implemented in the system code to bait the would-be interloper; they rely on the premise that more subtle recording/parameter checking techniques (the trap) will escape his attention. Many systems have inadequate penetrator entrapment and audit and surveillance mechanisms; others employ none of these mechanisms.
- Personnel Inefficiency—The operational management of operating systems (storage volume control, password assignment, virtual memory allocation, etc.) is dependent upon the care and competence of the people operating and maintaining the system. Redundancy checks, etc., to check the accuracy of the operational manager are lacking in most systems.
- Privity—This is a potential design weakness in many systems. Too many system programs and subsystems are given supervisor privileges to facilitate access to certain system tables. The more modules that operate in a supervisor state, the greater the chance for a serious penetration. Moreover, the hardware is inadequate, lacking descriptor based hardware, multiple execution states, and ring structures.[1]
- Program Confinement—Borrowed programs can act as Trojan Horses in that the borrower's files may be pilfered and his programs altered. Alternately, the borrower may copy a proprietary program for his own use. Lampson[19] has shown that borrowed programs may leak information through subtle "communication" channels.
- Prohibitions—System precautionary measures or prohibitions contained in User's Guides and other documentation may lead to system penetrations if the personnel responsible for operational management have not been fully cognizant of the prohibition's potential. For the penetrator, this weakness can become an easy exposure vehicle since the potential weaknesses are readily advertised by the manufacturer in the form of "Prohibitions", etc.
- Residue—Residue refers to unauthorized information that has been made available to the penetrator through poor housekeeping practices and system design carelessness. Trash baskets may be searched for log-on password or user identification. Trash also yields lists, notes, discarded teleprinter ribbons or platens, and data on paper, magnetic tapes, or disc packs which can be searched for sensitive information. In short, anything not erased or overwritten before being discarded is of potential use to a penetrator.
- Magnetic tape, disc space, and core residue can often be easily read and searched for sensitive information; temporary files and buffers are the most common sources.
- Security Design Omissions—If the security design is so complex, that many design decisions must be left to the implementor, the chances for a security design omission in the form of inadequate legality checking are enhanced.
- Shielding—In the absence of encryption techniques, computer rooms are shielded to counteract electromagnetic pickup.
- Threshold Values—Many internal surveillance checks in the form of threshold values leading to the entrapment of a penetrator are not present in many systems. For example, after a user has attempted to use a password that has failed N times, he should be locked from the system and the security officer should be notified.[12]
- Use of Test and Set—Many system designers were not cognizant in their design of an asynchronous attack potential by another processor. Hence, critical region parameters can be checked, then changed prior to the systems use by a concurrent process.
- Utilities—The operational management and control of system utilities is often neglected by the system manager. This area is fraught with Trojan Horse attack potential. Frequently, utility designers are not aware of security design issues. For example, inadequate boundary checking at compile time may allow a user to execute machine code disguised as data in a data area.

APPENDIX B—GENERIC OPERATING SYSTEM ATTACKS

- Asynchronous—Multiple processes overlapping each other's execution; one process attempts to change the parameters that the other process has legally checked but has not yet used.
- Browsing—A search by an authorized/unauthorized user for privileged or classified information in the computer system.
- Between Lines—To employ a special terminal tapped into the communication channels to affect "between

lines" entry to the system when a legitimate user is inactive but still holds the communications channel.

- Clandestine Code—The submission of "patch" code containing trap doors used to repair an operating system or utility program error, allowing the repairman to subsequently enter the computer system in an unauthorized fashion.
- Denial of Access—A program written to usurp a preponderant share of the system's physical resources or written to cause a system crash, the ultimate goal being the denial of legitimate users access to the computer system's resources in a timely manner.
- Error Inducement—A program written to cause errors within itself so that its subsequent state vector may contain information altered by the systems error recovery routine.
- Interacting Synchronized Processes—Processes that synchronize their execution through the system's synchronization primitives, sharing and passing information to each other through a common data base or subtle communication channel.[19]
- Line Disconnect—The attempt to gain access to another user's job subsequent to his disconnect but prior to hardware or software acknowledgment.
- Masquerade—To assume the identity of a legitimate user or process after having obtained proper identification through wiretapping or other means.
- "NAK" attack—This probabilistic attack exercises operating system weaknesses for not properly handling user generated asynchronous interrupts. The name comes from a user interrupt being commonly generated with the teletypewriter NAK (Negative Acknowledge) key. A system is often designed so that a user can interrupt a process, perform an operation, and then either return to continue the process or begin another. Poor designs often leave the system in an unprotected state during these times: partially written files left open, improper writing of a protection infraction message, etc. All possible situations of asynchronous interrupts are very difficult to analyze.[24]
- Operator Spoof—The attempt to circumvent installation management procedures by "spoofing" the computer system operator into an action that compromises the security of the system.
- Permutation Programming—A program written to process the total number of changes in position or order possible within a group. For example, to search for "new and useful" operation code.
- Piecewise Decomposition—A technique for cracking password and other Secret character strings by decomposing the string into its constituent characters and testing each in turn.
- Piggy Back—To employ a special terminal tapped into a communication channel to effect "piggy back" entry into the system by selective interception of communications between a user and the processor, and then releasing these with modifications or substituting entirely new messages while returning an "error" message.
- Trojan Horse—This term—first coined by Dan Edwards—refers to planting an entry point or "trap door" in the computer system to allow subsequent unauthorized access to the system. The Trojan Horse may be planted on a temporary or permanent basis. Trojan Horse also refers to any unexpected and malicious side effect, i.e., a program which executes a desired function correctly, but has illegitimate side effects.
- Unexpected Operations—To invoke seldom used system primitives or macros in an unusual manner taking advantage of the system's design and implementation weaknesses.
- Unexpected Parameters—To submit unusual or illegal parameters in a supervisor call attempting to circumvent or capitalize on the system's legality checking procedures.
- Wire Tapping—To cut in one or tap a communication channel to intercept a message.

A synthesizer of inductive assertions

by STEVEN M. GERMAN and BEN WEGBREIT
Xerox Palo Alto Research Center
Palo Alto, California

INTRODUCTION

Mechanical verification of program correctness is desirable and possible.[6] Given a program with complete, correct predicate specifications* on the input, output, and each loop, verification of the output predicate is a mechanical process (cf. References 5 and 15 for surveys).

It is necessary and natural for a programmer to supply input and output assertions. However, completely specified inductive assertions on loops are redundant. Writing such redundant loop assertions is a tedious and error-prone task for the programmer, and is therefore an obstacle to the practical use of program verifiers. This paper describes a prototype system, Vista, which provides assistance in synthesizing correct inductive assertions. Given only the source program and input/output assertions, it is able to generate a useful class of assertions totally automatically. For a larger class, it is able to extend partial inductive assertions to form complete assertions, from which it proves program correctness.**

There has been a substantial amount of work on program verifiers (e.g., References 3, 8, 11, 14, 16, 19) and the synthesis of inductive assertions (e.g., References 2, 4, 9, 12, 13, 18, 21). Vista is one of the first implementations of techniques for assertion synthesis. This is an interim report on the current state of the system. In the course of the implementation, several new techniques were conceived and a number of previously published methods[21] were better understood. The second section of this paper presents a set of examples showing the range of Vista and the techniques it employs. The third section discusses its implementation and current limitations. Throughout, we place particular emphasis on general methods which may be of interest to researchers constructing similar systems.

Before turning to specifics, three general comments seem in order.

(1) Even in the current prototype state, the power of individual modules is somewhat impressive. For example, Vista can generate the inductive assertions for the first seven examples used by King[14] given only the input/output assertions, while it can take the incomplete inductive assertion supplied for Example 9 in Reference 14 and extend it to the complete assertion from which it proves program correctness. Similarly, Vista can generate about half of the inductive assertions needed to verify the median-finding program Find.[10]

(2) The power of Vista is due in no small measure to the theorem prover it employs. This is a component of the program verifier Pivot[3] which is the work of L. P. Deutsch. Vista is built on Pivot, employing Pivot's internal routines to carry out theorem proving, derive logical consequences, maintain data bases, and a variety of similar tasks. Salient points of this relationship are discussed in the section on implementation.

(3) Vista is not yet a complete, integrated, or fully automatic system. It presently consists of a number of loosely-coupled specialist modules each of which applies a particular technique of assertion synthesis and operates completely automatically once invoked. The specialist modules communicate by storing and retrieving information from a global data base associated with the program being verified. A control program invokes the specialist modules as required. However, this is the weakest portion of the current system: While the control program is currently able to automatically handle simple cases such as the first seven examples of Reference 14, more complex examples are done in interactive mode in which the user invokes the specialist modules. The shortcoming of the control program range from straightforward issues in programming to some ill-understood problems of heuristic control. The overall control structure, both present and planned, is discussed in a later section.

EXAMPLES

Vista uses four principal methods to obtain inductive assertions:

(1) symbolic evaluation in a weak interpretation,

* The predicate specification is correct if each assertion is a valid consequence of the input assertion and processing done by the program. The specification is complete if each assertion can be proved true given that all immediately preceding assertions are true.

** Throughout this paper and throughout the system we tacitly assume that the program does conform to its specifications and that formal verification of this conformation is the only issue at hand. The case of incorrect programs which must be debugged from the specifications, or the case of correct programs in which some of the internal inductive assertions are wrong, present additional problems. We would contend that these problems are best addressed after the simpler case of correct programs is better understood.

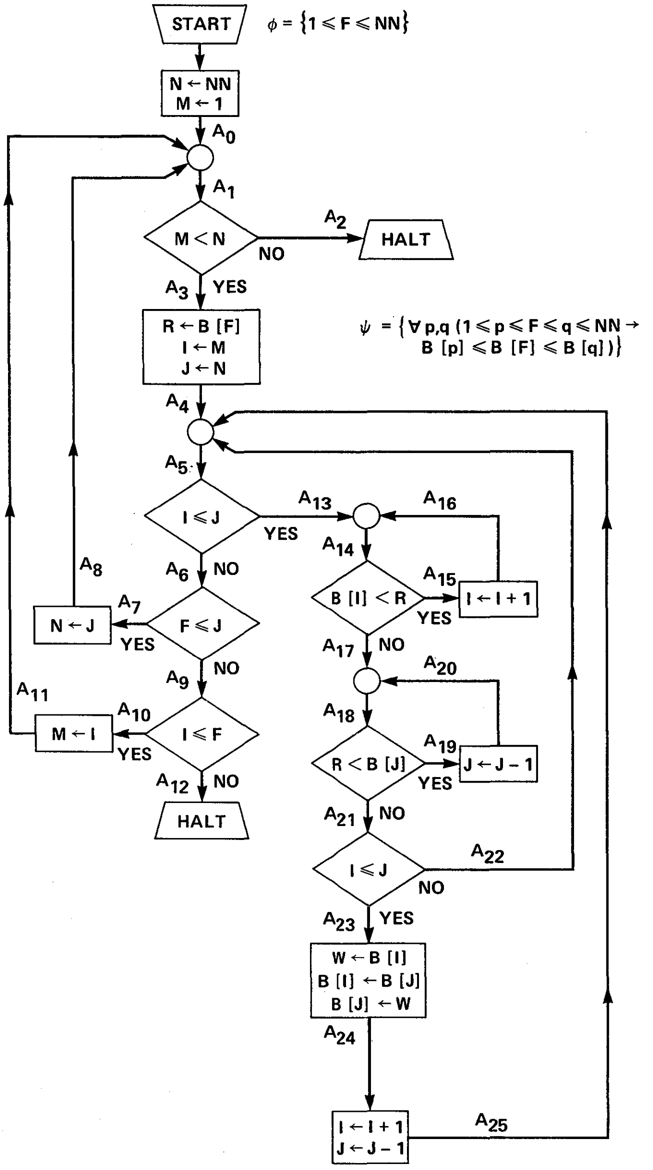


Figure 1—FIND

(2) combining output assertions with loop exit information to obtain trial loop assertions, and generalizing these where necessary,

(3) propagating valid assertions forward through the program, modifying them as required by the program transformations, producing assertions downstream,

(4) extracting information from proofs that fail in order to determine how assertions should be strengthened.

The first attempts to pick up obvious simple facts totally automatically. The others attempt to extend existing assertions—either to some other part of the program or by finding supporting lemmas needed to establish their correctness. We assume here that the programmer has supplied the essential ideas but may have omitted supporting details. The extent to which these details can be numerous, complex, and even subtle will be evident in the examples which follow.

We discuss these methods in turn, beginning each discussion with an explanation of the idea followed by an example to show how it is used in practice. Each example thus illustrates an invocation of one or more of Vista's specialist modules.

Notation. Throughout, a simple flowchart language is used. The input predicate is denoted by Φ; the output predicate by Ψ. Predicate P_i holds at arc A_i. To abbreviate a commonly used phrase, we say an assertion is *valid* if its verification conditions can be proved using assertions known to be correct.

Weak interpretation

In many programs, there is a useful amount of information that is obvious upon inspection. For example, suppose I and J are two integer variables. I is initialized to $J-1$ on entry to some loop, I is decremented on each path through the loop, J is unchanged, and the loop is exited when I turns negative. It follows that inside* the loop $O \leq I < J$. As a second example, consider what would occur if the body of the loop were modified by the addition of a conditionally executed branch containing the assignment $J \leftarrow I \leftarrow J-I$; in this case, $O \leq I \leq J$ would hold on the modified loop. As a third example, suppose the assignment $K \leftarrow B[I]$ was added to the body of the loop; in this case, it would follow that $K=B[I]$ and that assignments to $B[J+1]$ cannot affect this relation.

Weak interpretation attempts to derive simple facts of this sort; specifically, it considers only simple linear equalities or inequalities relating two variables. Its operation entails executing the computation steps symbolically, keeping track of the symbolic values of variables and of simple relations between them. When going repeatedly around a loop on which values are changed, it attempts to find a general expression which subsumes the particular cases. Thus, if an arc is reached with $I=N$ and then with $I=N-1$, the simple relation $I \leq N$ is tried. The outcome of weak interpretation is almost never enough to fully verify the program, but it does provide a set of valid relations which therefore need not be explicitly specified by the programmer.

For example, Figure 1 shows a program due to Hoare[10] for finding that element of an array $B[1:NN]$ whose value is F—th in order of magnitude. Weak interpretation results in the following valid assertions:

at arc A_1: $1 \leq M \leq F \leq N \leq NN$

at arc A_5: $M \leq I \;\&\; J \leq N \;\&\; 1 \leq M \leq F \leq N \leq NN$

at arc A_{21}: $B[J] \leq R \leq B[I] \;\&\; M \leq I \;\&\; J \leq N$
$\&\; 1 \leq M \leq F \leq N \leq NN$

at arc A_{24}: $B[I] \leq R \leq B[J] \;\&\; M \leq I \;\&\; J \leq N$
$\&\; 1 \leq M \leq F \leq N \leq NN$

* In this and in subsequent examples, we are deliberately informal, in order to present the main ideas as directly as possible. A precise statement would take into account the relative locations of the arc at which the assertion is formed and the points at which I is decremented.

Note the interdependence between the asignment $J \leftarrow N$ at A_3/A_4, the decrementing of J, the assertion $J \leq N$ at A_5, the test $F \leq J$, the assignment $N \leftarrow J$, and the assertion $F \leq N$ at A_1. Note also how $B[J] \leq R \leq B[I]$ becomes $B[I] \leq R \leq B[J]$ after the exchange.

Using loop exit tests and generalization

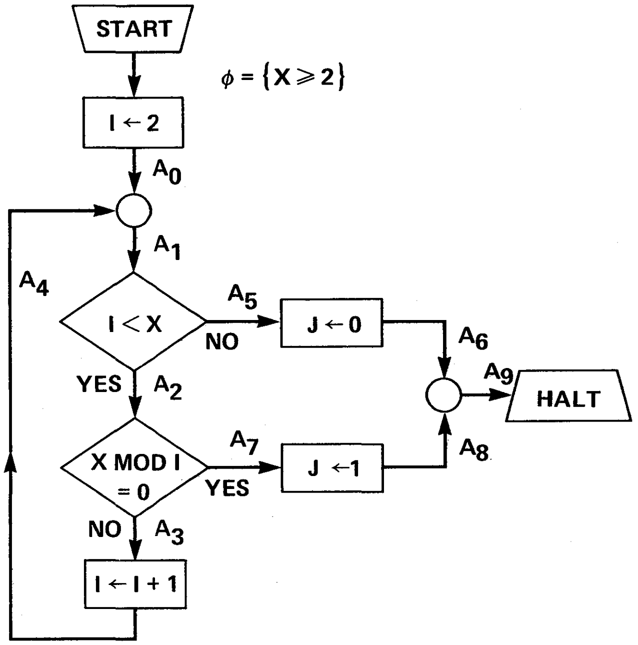
Suppose a loop is exited when some test D is true and that outside the loop some assertion P is to hold. Since P is to hold outside the loop, the assertion $\{D \rightarrow P\}$ must be true inside the loop, just before the exit test. $\{D \rightarrow P\}$ is the *weakest* inductive assertion at this point, in that any complete assertion must imply it. It may itself be a complete inductive assertion, in which case the theorem prover finds* it to be valid; or it may be incomplete, in which case it is found to be unprovable. If incomplete, it must be strengthened. Often, the forms of P and D suggest appropriate generalizations.** Suppose P asserts that some predicate Q is true of each j in the range $1 \leq j \leq N$ and D asserts that the loop is exited when $I \geq N$; one might hypothesize that inside the loop Q is true of each j in the sub-range from 1 to I, e.g., because the loop is counting up on I. This generalization, $\forall j(1 \leq j \leq I \rightarrow Q(j))$ is therefore tried as an inductive assertion.

As an example, consider the program in Figure 2 which tests whether X is prime. It sets the flag J to 0 if X is prime and to 1 otherwise. Vista can verify the program given only the input, Φ, and output, Ψ, assertions as shown. Combining Ψ with the loop exit information and simplifying, it obtains the weakest loop assertion: $\{I<X \vee \forall k(2 \leq k<X \rightarrow \mathrm{X\ MOD}\ k \neq 0)\}$. This is tested for validity, found to be unprovable, and then strengthened. The result, $\{\forall k(2 \leq k< I \rightarrow \mathrm{X\ MOD}\ k \neq 0\}$, is found to be a valid assertion. Further, it validates the output assertion, thus proving the program correct.

Predicate propagation

Whenever an assertion is known to be valid, it is useful to propagate it forward in the program, deriving the strongest consequences of the assertion downstream. Note that the verification condition[6] for a path deals only with the path's processing and the assertions at the head and end of the path, totally ignoring the rest of the program. Consequences of a valid predicate propagated downstream to a cutpoint may add new information which is absent in the inductive assertion at that cutpoint. On the path for which the cutpoint is the head, the new information may render valid a verification condition which would be invalid without the new information. Predicate propagation has three levels of sophistication.

Assertions which continue to hold, e.g., because they involve variables unchanged on some path, are discovered. This is primarily a notational convenience: an assertion which holds over some large program region and is needed as a lemma in many loop verifications need be stated only once, at the head of the region.



$$\psi = \{[J=0 \rightarrow \forall k\ (2 \leqslant k < X \rightarrow X\ \mathrm{MOD}\ k \neq 0)] \wedge [J=1 \rightarrow X\ \mathrm{MOD}\ I = 0 \wedge I < X]\}$$

Figure 2—Testing for prime

Assertions are modified on passing through decisions and assignments to produce their consequences. On passing through a decision D in the Yes direction the clause D is added, and conversly for the No direction. On passing through an assignment $V \leftarrow E$, the clause $V=E$ is added and all uses of the old value of V are systematically eliminated. For example, if $B[J] \leq R$ is an assertion, then after passing through the test $R \leq S$, we have $\{B[J] \leq R \leq S\}$; after the assignment $R \leftarrow C$, we have $\{B[J] \leq S\ \&\ R=C\}$; further, after the assignment $J \leftarrow J-1$, we have $\{B[J+1] \leq S\ \&\ R=C.\}$

When a junction is encountered, the assertion becomes a *trial* assertion for testing on the junction output arc. If it is valid there, a possibly useful new fact has been discovered. It may be unprovable, in which case a generally weaker assertion is formed by taking the disjunction of known assertions on all inputs to the junction and tested as a new trial assertion on the junction output arc.

A complete example of predicate propagation may be helpful. Figure 3 shows a simple sort program, Example 9 of Reference 14. Arc A_5 is tagged by hand with an incomplete assertion: that the portion of the array from $B[1]$ to $B[I]$ is sorted and that X is no larger than any element in the portion of B from $B[I]$ to $B[J-1]$. A complete assertion

* Throughout this paper, we assume that the theorem prover is able to confirm all valid theorems presented to it by Vista. The theorem prover is, of course, not complete over the integers with exponentiation—that is impossible by undecidability arguments. However, in practice, it turns out that the theorem prover can handle all the theorems we require.

** This is discussed in more detail in References 7 and 21.

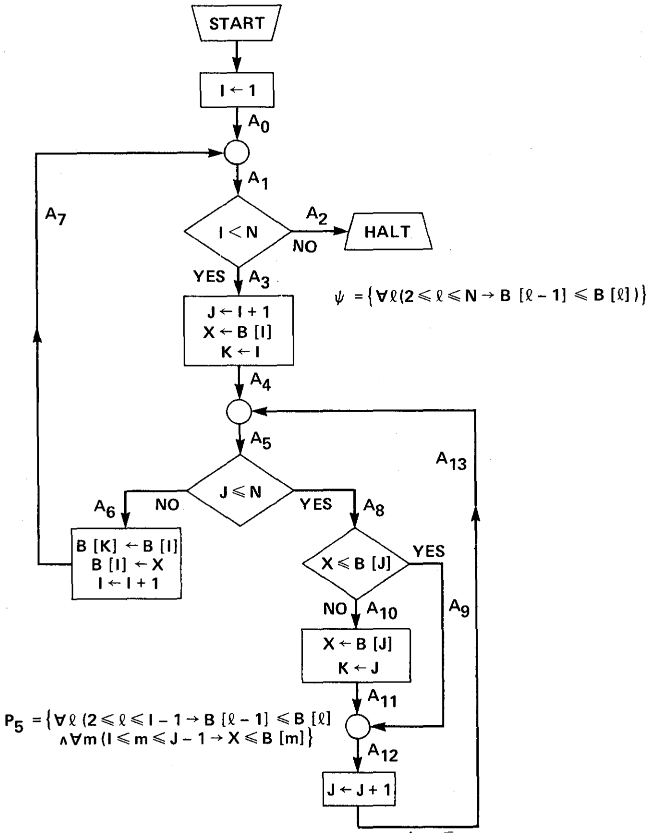


Figure 3—Sort by successively finding the smallest

would also include the key fact that no element in the sorted portion exceeds any element in the unsorted portion; lacking this, verification cannot proceed. Vista discovers the incompleteness when checking the inductive assertions for validity. The assertion that $B[1{:}I]$ is sorted is found to be initially true but not provable for subsequent passes around the loop $A_5A_6\ldots A_4A_5$. However, the second assertion, $\forall m(I\leq m\leq J-1 \rightarrow X\leq B[m])$ is found to be valid and is so recorded; Vista then propagates this valid assertion forward. The path which turns out to be interesting is $A_5A_6A_7A_1A_3A_4A_5$. Going along it produces the valid assertion at arc A_4: $\{I=1 \vee \forall m(I-1\leq m\leq N \rightarrow B[I-1]\leq B[m])\}$ which may be read as: either this is the first time through the loop or $B[I-1]$ is the smallest element in the subarray between $B[I-1]$ and $B[N]$. This is the missing key fact. It is propagated forward to arc A_5 where it becomes a trial assertion, is checked for validity, and is found to be valid. Using weak interpretation as discussed previously, Vista also generates the additional assertions at A_5: $\{I\leq K<J\leq N+1$ & $I<N$ & $X=B[K]\}$. With these two sets of auxiliary assertions, the output predicate is then validated, thus proving the program correct.

Extracting information from unsuccessful proofs

When proving mathematical theorems, if some approach fails it is often useful to analyze the cause of failure and modify the approach to fix up the fault. This idea carries over to program verification. Suppose a loop* contains a trial assertion P—either specified by the programmer or generated as a trial assertion by one of the methods discussed previously—which cannot be verified around the loop. More precisely, let D be the set of conjuncts added because of decisions and let $F(\xi)$ be the transformation to the state vector ξ caused by loop assignments; loop verification requires that $\{\forall\xi(P(\xi)\ \&\ D(\xi) \rightarrow P(F(\xi)))\}$ and this may be unprovable. Under the assumption that P is correct but incomplete, it follows that $P(F(\xi))$ must be true; hence, it is necessary to find an additional condition C such that $\{C(\xi)\ \&\ P(\xi)\ \&\ D(\xi)\}$ imply $\{C(F(\xi))\ \&\ P(F(\xi))\}$. Often it turns out that the verification condition fails only for certain identifiable cases and these cases may be used to construct the additional condition C. An example will best explain this.

Consider the program of Figure 4 which sorts an array B by straight insertion. At the start of the J-th pass, the

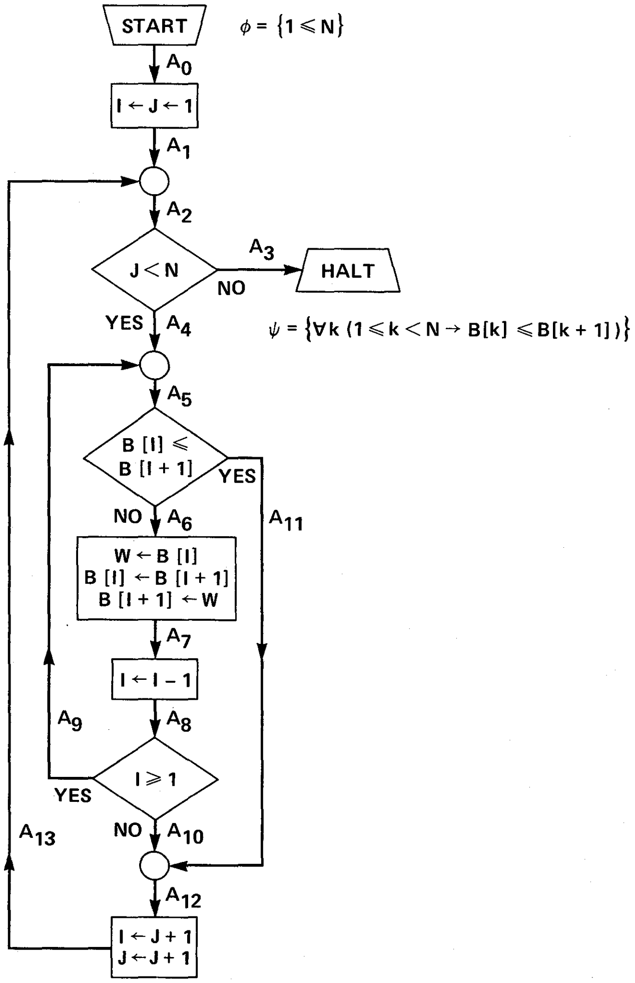


Figure 4—Straight insertion sort

* We discuss the application of assertion correction to a closed path; however, the idea can be used on any path.

subarray $B[1{:}J]$ has been sorted; the J-th pass inserts the element $B[J+1]$ in its correct position within the sorted subarray, thus sorting the subarray $B[1{:}J+1]$. Suppose Φ and Ψ are supplied as shown, along with the partial inductive assertion on arc A_5, that $P_5=\{\forall k(1\leq k\leq J \ \& \ k\neq I \rightarrow B[k]\leq B[k+1])\}$. Weak interpretation as discussed earlier adds $\{1\leq I\leq J<N\}$. Vista attempts to prove P_5 is valid and finds that the proof fails for only one special case on the loop $A_5A_6A_7A_8A_9A_5$: if $B[I]>B[I+1]$ then after the exchange of $B[I]$ with $B[I+1]$ there is no way to show that $B[k]\leq B[k+1]$ when $k=I+1$. This would be true if and only if $B[I]\leq B[I+2]$ before the exchange and this has not yet been asserted. However, if P_5 is correct, this *must* be valid. Further, since $1\leq k\leq J$ it follows that $I+1=k\leq J$, i.e., $I<J$. This fact is included as part of the case information, resulting in a condition. Vista's analysis continues with path $A_5A_6A_7A_8A_{10}A_{12}A_{13}A_2$ on which the proof fails in two cases, yielding other conditions. The conjunction of the conditions simplifies to $\{I<J \ \& \ B[I]>B[I+1]\rightarrow B[I]\leq B[I+2]\}$. This may be read as: even if $B[I]$ is out of order, it is still smaller than $B[I+2]$. With this conjoined, P_5 is valid and validates Ψ, thus proving the program correct.

IMPLEMENTATION

In the interest of brevity, we confine our discussion to a few major points of the implementation. Further detail, treatment of other points and discussions of certain aspects of the overall organization can be found in References 7, 20 and 21. In particular, the reader will find there discussions of the theory and logical basis of the techniques presented below.

System structure

The major modules of Vista are (1) the weak interpreter, (2) trial generalization generator, (3) predicate propagator and (4) assertion correction and predicate initialization mechanism. Thus the example in the section *weak interpretation* represents a single call to the weak interpreter. The section *using loop exit tests* is handled by the control program, using the predicate initialization mechanism and the generalization generator. The section *predicate propagation* represents manual invocation of the weak interpreter followed by the predicate propagation mechanism. The section *extracting information from unsuccessful proofs* describes a manual invocation of the assertion corrector. It is indicative of the current status of Vista that when modules are invoked manually on the program in Figure 4. the complete assertions as both A_2 and A_5 are produced without any programmer-supplied inductive assertions, but that the control program is currently unable to duplicate this.

The current control program is applicable only to simple programs. We now describe a more general control strategy, which at present, would have to be executed interactively. The first step in assertion synthesis is to evaluate the entire program with the weak interpreter. This is a good start, because the weak interpreter produces assertions of guaranteed correctness that are useful during later phases of assertion synthesis and it does not itself require information other than the input assertion. Next, the strategy is to work on one cutpoint at a time, starting from those closest to the output assertion and working backwards. At each cutpoint, we attempt to convert the information available from weak interpretation and programmer-supplied assertions into a complete assertion. The predicate initialization mechanism is used to generate the weakest possible assertion. Then, as in the example involving loop exit tests, a number of possible generalizations of the weakest assertion are formed and tested for validity. Note that in multiple-loop programs, it is not possible to test for correctness until all cutpoints have complete assertions; at intermediate phases of assertion synthesis, loops without assertions are approximated by finite expansions of their bodies. If none of the trial assertions are correct and complete, the theorem prover is used to discard non-invariant clauses and form the strongest correct but incomplete assertion from the trial assertions. This predicate is propagated along closed paths back to the cutpoint. The result of predicate propagation is tested for new invariants and, if any are discovered, new generalizations are formed and tested. The discovery of a new invariant is a valuable gain in information. Therefore, we employ the new fact as quickly as possible. Otherwise the assertion correction mechanism is used to produce a new predicate to generalize and test. Synthesis proceeds in this fashion until a complete assertion is found.

Weak interpretation and predicate propagation

Weak interpretation and predicate propagation both attempt to derive assertions directly from the program structure, using similar processing steps. We discuss predicate propagation first and then explain how weak interpretation differs.

Predicate propagation starts with a known valid assertion P_0. Following the flowchart, processing at each node transforms a predicate P at the node input arc to obtain consequent predicate(s) at the output arc(s). Predicates are always expressed using current values of variables.

A decision node D conjoins to P the clauses D and $\sim D$ on its Yes and No output arcs respectively. It is frequently the case that the new clause could, if we wished, be combined with existing clauses of P to generate additional clauses. For example, after adding $S\leq T$ to $\{R\leq S \ \& \ T\leq U\}$ we could obtain the logically valid additional clauses $\{R\leq T \ \& \ R\leq U \ \& \ S\leq U\}$. The set of such additional clauses can, in general, be large—filled with redundant information. Hence, it is not a good practice to carry out the expansion. Instead, only simplification is performed. For example, if $P_1 = \{A\leq B \ \& \ C=D+1 \ \& \ F[B]\leq E\}$ is the predicate on the input arc, then the predicate on the Yes output arc of the decision $B=C$ is $P_2=\{A\leq B \ \& \ C=D+1 \ \& \ F[B]\leq E \ \& \ B=C\}$.

Next, consider an assignment $X\leftarrow G(X, Y)$ where $R(X, Y)$ is the known predicate before the assignment. The *strongest* *predicate* after the assignment is: $\{\exists X_0(R(X_0, Y) \;\&\; X=G(X_0, Y))\}$. However, this is too strong. Since it contains a new variable X_0 and since predicates are always expressed using current values of variables, this is unsuited to further propagation. Instead of using it directly, an attempt is made to find a logical consequent in which X_0 does not appear. This may be done in one of two ways depending on the form of the right hand side of the assignment:

(1) An invertible assignment is one which may be inverted to obtain the old value of the assigned variable as a function of its new value, i.e., $X_0=H(X, Y)$ for some appropriate function H. Using this, X_0 may be directly eliminated from the strongest predicate, resulting in $\{R(H(X, Y), Y)\}$. For example, consider the above P_2 followed by the assignment $B \leftarrow B+2 \cdot E$. This may be inverted to obtain the old value of B as a function of its new value, i.e., $B_{\text{old}}=B_{\text{new}}-2 \cdot E$. Substituting the right hand side, $B-2 \cdot E$, for the left, B, in P_2 expreses the relations using the new value of B. Thus, the predicate on the output arc of the assignment is $P_3=\{A \leq B-2 \cdot E \;\&\; C=D+1 \;\&\; F[B-2 \cdot E] \leq E \;\&\; B-2 \cdot E=C\}$.

(2) A non-invertible assignment is more complex. Clauses which depend on the changed variable are deleted. However, the consequences of these clauses may not depend on the changed variable and these consequences may be invariants; hence, these consequences are derived and added to the data base before deletion. For example, if the above P_2 were followed by $B \leftarrow F[D]$, then the clauses $A \leq B$, $F[B] \leq E$, and $B=C$ would not be valid on the output arc; however, $A \leq C$, and $F[C] \leq E$ which are consequences *are* still valid, so these are generated and kept. As a final step, an equality expressing the new value of the changed variable is conjoined. For this example, the result is $P_4=\{A \leq C \;\&\; F[C] \leq E \;\&\; C=D+1 \;\&\; B=F[D]\}$.

A junction node is the critical step in predicate propagation since it causes confluence of control and with it the need to find an assertion on the output that is valid for *each* of several paths leading to it. Consider a junction with n input arcs on which prior processing has resulted in the valid assertions $P_1, \ldots, P_n$. Let P_{n+1} be the valid assertion obtained on the output arc from prior processing (e.g., the starting predicate).

From $P_1, \ldots, P_{n+1}$, a new assertion on the output arc, P_{out}, is obtained. Each P_i is written as a set of clauses $P_i=C_i^1 \;\&\; C_i^2 \;\&\; \ldots \;\&\; C_i^{k_i}$ and each conjunct C_i^j is tested on the implication $(P_1 \vee \ldots \vee P_{n+1} \rightarrow C_i^j)$. If this is provable, then C_i^j is a conjunct of P_{out}. Clauses which fail this simple test are tried as loop assertions and tested for validity on entry to and around the loop. Clauses which prove to be valid are added to P_{out}. Let P_i' be the residue after removing from P_i the clauses thus found to be in P_{out}. To propagate this residue, $(P_i' \vee \ldots \vee P_{n+1}')$ is formed and added as a conjunct of P_{out}. Finally, the new output assertion, P_{out}, is compared with the previous output assertion, P_{n+1}. Any clause in P_{out} not in P_{n+1} represents a new valid assertion and may be propagated forward.

Weak interpretation differs from predicate propagation in two respects. First, there is no known valid assertion to start with. We take as starting predicate P_0 on a loop the disjunction of the results of propagating the input assertion along all acyclic paths from the start node to the entrance of the loop. This is usually *not* a loop invariant; however, it can be used as a trial predicate and from it the weak interpreter attempts to construct a suitable generalization which *is* an invariant. Only simple linear relations between two variables are considered; hence, finding suitable generalizations by using domain-specific heuristics is often possible.* Secondly, the implementation differs in that weak interpretation requires only very simple deductions, rather than the full power of the theorem prover required for predicate propagation.

With the understanding that only those clauses expressing simple linear relations are carried, decision and assignment nodes are treated as in predicate propagation. When processing a junction, the first step is as in predicate propagation: Each conjunct C_i^j of each P_i is tested in $(P_1 \vee \ldots \vee P_{n+1} \rightarrow C_i^j)$ and successful conjuncts are included in the output predicate P_{out}. Generalization occurs when handling the conjuncts which fail this test. For example, suppose $n=2$, P_1 includes $I=J$, and P_2 includes $I=J+1$. The disjunction $\{I=J \vee I=J+1\}$ would be a logical consequence but probably not a loop invariant. Weak interpretation forms $\{I \leq J+1 \;\&\; I \geq J\}$ which is logically equivalent but is in a form more suggestive of generalization. This is in conjunctive form and is propagated around the loop again. If one of these conjuncts is an invariant then it will be implied by the result of further propagation and so it is kept, while the non-invariant conjunct is not thus implied and so is dropped. In this way, special cases comprising the starting trial predicate are progressively replaced with trial generalizations, some of which fail and are dropped while others are invariants and remain.

Constructing and correcting inductive assertions

Vista's abilities to construct inductive assertions from output predicates and to correct assertions from proofs that fail both depend heavily on a close working relationship between the trial assertion generator and Pivot, the theorem prover. In particular, Pivot's theorem proving process is structured to leave a record of the proof that is meaningful in relation to the program. At all major steps, Pivot records what actions it is taking and its reasons for taking them. Thus, when a proof fails, a body of useful information is produced for analysis by the trial assertion generator.

Vista's predicate *construction* mechanism forms the initial trial assertion inside a loop. Assume that a predicate P_0 must be shown to hold outside a loop, that P_I is known to be true inside (e.g., from weak interpretation), and that C is the computation in exiting the loop. Vista proceeds by asking Pivot to prove $(P_I \;\&\; C \rightarrow P_0)$. Usually, P_I will be insufficient

* In order to keep this discussion reasonably short, we are somewhat imprecise in several places. In particular, the method is only correct under suitable restrictions. See [20, 21] for a discussion of these restrictions and proofs of correctness.

for the proof to succeed; however, in the process of theorem proving, Pivot will find (implicitly) and simplify the additional facts that are needed inside the loop. For example, consider a loop with exit test $D=\{A[I]\leq Y\}$ for which the assertion $P_0=\{\exists v(Y\leq v\leq A[I])\ \&\ Q(v)\}$ must hold after exit. Assume the path that exits from the loop contains the assignment $A[J]\leftarrow X$. Then if it is known by weak interpretation that $\{I=J\}$ inside the loop, Vista will form the trial assertion $\{A[I]\leq Y\rightarrow[\exists v(Y\leq v\leq X)\ \&\ Q(v)]\}$ because $\{A[I]\leq Y\}$ is the exit condition and after the assignment $A[J]\leftarrow X$ the value of $A[I]$ in P_0 will be X.

When *correcting* assertions after proofs which fail, Vista's methods are similar but include special procedures for proofs involving simple arithmetic relations. One very useful method is to deduce a new invariant equality when the proof of a trial equality fails. For example, if Vista is considering the trial assertion $\{X=A^N\ \&\ N\geq 0\ \&\ M\geq 0\}$ on a loop with the assignments $X\leftarrow X\cdot B$, $N\leftarrow N+M$, then the clauses $N\geq 0$, $M\geq 0$ may be proved invariant, while $X=A^N$ will be found to be unprovable. Vista substitutes the values of variables after following the path around the loop into $X=A^N$, producing $X\cdot B=A^{N+M}$. It then uses a simple equation solver to find that this new equality would be satisfied if $A^N\cdot B=A^{N+M}$. If this new condition is satisfied upon entrance to the loop, it will be an invariant, and will allow $X=A^N$ to be proven invariant. Hence, it is conjoined to the trial assertion and a new invariant assertion is produced.

Vista's method for deducing a new invariant equality takes the original clause that could not be proven $(e=0)$, and the value of the clause at the ene of the path $(e'=0)$ and gives them to a simple equation solver which forms new equations by eliminating variables. If either equation allows direct solution for a variable, the value is substituted in the other equation yielding a new equality. A variable V which cannot be solved directly may sometimes be eliminated by finding a pair of arithmetic terms α, β such that all terms in $\alpha e+\beta e'=0$ which contain V are cancelled. All new equations formed by elimination of one or more variables are then tested for validity.

We now consider the example of an earlier section in more detail to show how Pivot is used to determine exactly why a trial assertion is insufficient. The path around the inner loop of Figure 4 switches $B[I]$ with $B[I+1]$ and decrements I. Consider the steps involved in trying to validate the partial inductive assertion $\{\forall k(1\leq k\leq J\ \&\ k\neq I\rightarrow B[k]\leq B[k+1])\}$, specifically in trying to show that if the assertion is true on one pass through the inner loop then it will still be true on the next pass. Pivot forms cases by considering the possible values for k. For values of k other than $I-1$, I, and $I+1$, $B[k]$ and $B[k+1]$ are unchanged. With $k=I-1$ the assertion will be vacuously satisfied on the next pass through the loop. When $k=I$ the goal is established directly. The only remaining value of k is $I+1$. Pivot decides to prove the path by forming two cases: $k\neq I+1$ and $k=I+1$. The proof of the first case succeeds. The second proof fails, leaving a record that when $B[I]>B[I+1]$ and $I\geq 2$ (so that the inner loop will be followed) and $1\leq k\leq J$ (so that k is the domain of the goal quantifier) and $k=I+1$ (for a case restriction) then the goal clause $B[I]\leq B[I+2]$ cannot be established.

Vista's assertion correcting mechanism examines Pivot's records and finds that the goal was created by instantiating the quantified trial assertion with $k=I+1$. Vista then forms a new universally quantified assertion in which the body is the original goal clause, in which k is the bound variable and in which the domain is the conjunction of the old domain and case restrictions on k. This quantifier is then simplified by Pivot's expression routines. Since the bound variable can be eliminated, the new quantifier reduces to $I<J\rightarrow B[I]\leq B[I+2]$. Finally, Vista retrieves the decision information $I\geq 2$ and $B[I]>B[I+1]$ and forms the additional condition $\{I\geq 2\ \&\ B[I]>B[I+1]\ \&\ I<J\rightarrow B[I]\leq B[I+2]\}$. Analysis of the other path that fails produces similar results applicable to that path. Together with the loop-path result, these simplify to produce the new assertion $\{I\geq 1\ \&\ I<J\ \&\ B[I]>B[I+1]\rightarrow B[I]\leq B[I+2]\}$ and allow the program to be proven correct.

CONCLUSION

Where do we go from here? Given the current state of Vista, what steps should be taken next and what prospects can be seen for production program verifiers in day-to-day use carrying out assertion synthesis?

There is currently substantial research, ongoing at various laboratories, investigating other methods for synthesizing inductive assertions. Particularly promising is the use of difference equations relating the values of program loop variables, whose solution may be used to obtain invariants.[4,12,13] We have implemented a package which finds inductive assertions in this way and are currently integrating it into Vista. Symbolic evaluation of the program with simplification and pattern matching has been studied[2,9,18] as another means for finding invariants. The system of Boyer and Moore[1] for proving theorems about pure Lisp functions tries to generalize theorems containing common subexpressions; in so doing, it uses methods related to the idea of strengthening partially specified assertions.

We suggest that an intermediate-to-expert level of competence is necessary if a verifier is to be of real use in verifying production programs. In particular, the assertions supplied by the programmer should be no more extensive than those used to explain the workings of a program to an experienced programmer. To achieve this level of competence, all the above methods are required. To a large extent, these methods are both non-overlapping and complementary. For example: difference equations are a well-understood means for obtaining equality invariants, but relatively useless for inequalities and disequalities (e.g., $a\neq b$); weak interpretation is well-suited for simple relations (both equalities and inequalities) but unable to produce bounded universally quantified assertions; examining the causes of failure in order to patch up an assertion which fails is very powerful, but only for trial assertions which are sufficiently close to the final invariant. Further, it is frequently the case that a relatively simple fact found by one method unblocks some other methods, allowing a powerful line of attack to proceed. For these reasons, it seems most profitable to couple several good limited approaches of problem solution, rather than attempting to rely exclusively on one.

It must be emphasized that research in assertion synthesis is still in its infancy. Assertion synthesis at the level we believe desirable is still a distant goal. Programs of realistic size and complexity present a range of problems for which we have, as yet, no good solutions. Programs containing errors and the attendant problems of reconciling programs with their specifications offer further, and still harder, challenges.

Two related issues requiring further investigation are worth noting:

(1) The language for specifying assertions should be improved to facilitate specification of necessary assertions by the programmer (e.g., c.f. Reference 17). With the exception of Reference 1, there has been little work involving verification of programs containing assertions with programmer-defined recursive predicates. Also, it is difficult to express assertions about programs which manipulate list structure destructively. Studies in these area are being carried out and progress may be expected. It is then necessary to discover associated techniques for understanding, generalizing, and synthesizing assertions in these richer linguistic spaces.

(2) The theorem prover remains a fundamental module of any assertion synthesizing system. Improvements in domain-oriented theorem proving are therefore essential. A particular need which arises in assertion synthesis is the ability to efficiently check the validity of a formula and a number of slightly varied formulas (e.g., obtained by the deletion of conjuncts in the hypothesis of an implication). One would like a theorem prover to be able to simply extend, where possible, its proof of one formula when trying to prove a variant.

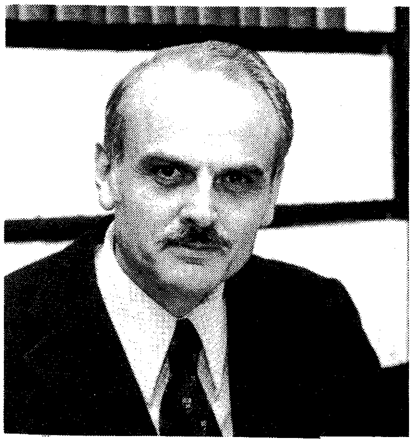
Substantial progress has been made in recent years toward the construction of production program verifiers; much remains to be done. The synthesis of inductive assertions is only one component, but an important one. Vista demonstrates that assertion synthesis is possible and can achieve significant performance.

ACKNOWLEDGMENTS

We wish to thank L. Peter Deutsch for his assistance in the use of his program verifier Pivot.

REFERENCES

1. Boyer, R. and J. Moore, "Proving Theorems about LISP functions," *Proc. 3rd Internat. Joint Conf. on Artificial Intelligence*, Aug. 1973, pp. 486-493.
2. Cooper, D. C., "Programs for Mechanical Program Verification," in *Machine Intelligence 6*, B. Meltzer and D. Michie (Eds.), American Elsevier, New York, 1971, pp. 43-59.
3. Deutsch, L. P., *An Interactive Program Verifier*, Ph.D. Thesis, Dept. of Computer Science, U. of California at Berkeley, 1973.
4. Elspas, B., *The Semiautomatic Generaton of Inductive Assertions for Proving Program Correctness*, SRI Project 2686, Stanford Research Institute, July 1974.
5. Elspas, B., K. L. Levitt, R. J. Waldinger, and A. Waksman, "An Assessment of Techniques for Proving Programs Correct," *Computing Surveys*, 4, 2, June 1972, pp. 97-147.
6. Floyd, R., "Assigning Meanings to Programs," in *Proc. of a Symposium in Applied Mathematics*, Vol. 19, J. T. Schwartz (Ed.) AMS, 1967, pp. 19-32.
7. German, S. M., *A Program Verifier that Generates Inductive Assertions*, Technical Report TR 19-74, Center for Research in Computing Technology, Harvard U., Aug. 1974.
8. Good, D. I., "Provable Programs and Processors," *AFIPS Conference Proceedings*, Vol. 43, *1974 National Computer Conf.*, pp. 357-363.
9. Greif, I. and R. Waldinger, "A More Mechanical Heuristic Approach to Program Verification," in *Internat. Symp. on Programming*, Paris, April, 1974, pp. 83-90.
10. Hoare, C. A. R., Proof of a Program: FIND," *C. ACM*, 14, 1, Jan. 1971, pp. 39-45.
11. Igarashi, S., R. L. London, and D. C. Luckham, *Automatic Program Verification I: AIM-200*, CS-73-365, Computer Science Dept., Stanford U., May 1973.
12. Katz, S. M. and Z. Manna, "A Heuristic Approach to Program Verification," *Proc. 3rd Internat. Joint Conf. on Artificial Intell.*, Aug. 1973, pp. 500-512.
13. Katz, S. and Z. Manna, *Logical Analysis of Programs*, Dept. of Applied Mathematics, Weizmann Inst. of Science, Rehovot, Israel, July 1974.
14. King, J., *A Program Verifier*, Ph.D. Thesis, Computer Science Dept., Carnegie-Mellon U., 1969.
15. London, R. L., "The Current State of Proving Programs Correct," *Proc. ACM 25th Ann. Conf.*, 1972, pp. 39-46.
16. London, R. L. and D. R. Musser, "The Application of a Symbolic Mathematical System to Program Verification," to appear in *ACM 74*.
17. Marmier, E., "A program verifier for PASCAL," *Proc. IFIP Congress 74*, North-Holland Publishing Co., pp. 177-181.
18. Moriconi, M., *Semiautomatic Synthesis of Inductive Predicates*, ATP-16, Depts. of Mathematics and Computer Sciences, U. of Texas at Austin, June 1974.
19. Waldinger, R. J. and K. N. Levitt, *Reasoning About Programs*, Artificial Intell. Center, Technical Note 86, Stanford Research Inst., Oct. 1973.
20. Wegbreit, B., *Property Extraction in Well-Founded Property Sets*, Center for Research in Computing Technology, Harvard U., Feb. 1973.
21. Wegbreit, B., "The Synthesis of Loop Predicates," *C. ACM*, 17, 2 (Feb. 1974), pp. 102-112.



Area Director:
E. F. Codd
IBM Research Laboratory
San Jose, California

Data base management

Data base management is in a state of ferment due to the emergence within the past few years of many new requirements. Amongst these requirements are the need to make application programs and terminal activities much more independent of the internal representation of data in storage, and the need to support:

1. many different kinds of end users at terminals (some interactions being of unpredictable scope and complexity);
2. greatly enhanced data security and privacy;
3. increased dynamic sharing of data (including concurrent update and enquiry);
4. networks of mutually remote data bases (including very high level data sublanguages for low bandwidth communication of requests).

In the data base management sessions of this year's conference there is a strong emphasis on *relational data base management.* There are three reasons for this: first, this approach appears to be the most advanced in attempting to meet all these new requirements; second, the overwhelming majority of data management papers submitted to NCC 75 were concerned with implementing this approach; third, there has been almost no exposure of this approach before in this forum, even though the ideas have gained a solid acceptance in Europe and have spawned a pronounced surge in data base oriented research in numerous universities in North America (notably M.I.T., Toronto, Berkeley, Florida, and Utah).

What then are the distinctive features of this approach? This topic will be covered by Christopher J. Date in a tutorial to be presented in the first half of session L1. Mr. Date is the author of "An Introduction to Database Systems" just published by Addison Wesley. This book contains a thoughtful and very clear comparison of the hierarchic, network, and relational approaches. The second half of session L1 consists of a panel discussion on the two questions:

1. What are the major problems in implementing relational data base management systems?

2. Is there any necessary loss of performance if performance-oriented access paths are known to the system but not to the application programmer?

Each panel member has been directly involved in implementing a relational data base management system.

The remaining sessions L2 through L7 are based entirely upon submitted papers. Session L2 provides a striking contrast between two data base machines: one based upon hierarchic data structures, the other based upon non-hierarchic relations. Three more relational implementations are described in session L3. These implementations differ markedly in scope and style.

Session L4 deals with relational data base technology. A data base management system based upon the relational model must support a user view that is devoid of performance considerations. Since the problem of selecting efficient retrieval algorithms is removed from the user, this burden must fall upon the system itself. One of the papers in session L4 describes an unusual technique for efficient interpretation of data selection expressions which involve inter-entry relationships. A second paper in L4 introduces an important unifying mechanism for the services of concurrency locking, authorization, and support of multiple tabular views of data.

Session L5 is concerned with the human factors aspects of query languages and with attempts to develop objective experiments for evaluating these languages. The first paper in this session introduces a novel approach to querying a relational data base using a terminal display and employing a technique of specification by example. Then follows a psychological study in which human subjects (all non-programmers) were taught this technique and tested for speed and accuracy in formulating sample queries of various complexities given informally in English. The third paper reports on a human factors experiment designed to evaluate and compare two high level data base query languages in use by a sample of programmers and a sample of non-programmers. These research efforts can be expected to trigger many similar investigations in the future.

Session L6 deals with more traditional topics in data management. The three papers have as their topics data compression (a useful survey is provided), binary search trees (which are important for directories and certain kinds of indexes), and performance evaluation (a large scale simulation model is described).

The final session L7 deals with distributed data bases and two important application areas—the medical field and urban management. The paper on distributed data bases provides a framework in which to tackle the problem of allocating files and programs to the nodes of the network. A second paper describes a clinical data base system that attempts to cope with the multiple user and data entry problems. The urban management paper suggests that totally integrated on-line data bases for urban problem solving and decision making are not yet practical, but that the employment of data files extracted from a common base for various specialized uses represents a feasible approach.

RAP—An associative processor for data base management

by E. A. OZKARAHAN, S. A. SCHUSTER and K. C. SMITH
University of Toronto
Toronto, Ontario

INTRODUCTION

Problems with DBMS on conventional machines

Recent concepts in data base management systems (DBMS) necessitate making the logical view and the physical representation of data distinct from each other. Currently, this requirement has to be realized in the environment of conventional Von Neumann architecture. This creates the need for several levels of indirection for mapping one structure into the other. Also, efficient search mechanisms are needed to handle large data bases within concurrent processing and on-line response limits. The implementation of these requirements results in software complexities and inefficiencies in the following way. Pointer mechanisms for mapping structures and providing fast access paths have to be implemented by software and data. These pointers are extra data requiring extensive overhead in storage, access time, and maintenance.

Recent approaches to hardware support for nonnumeric processing

The limitations of conventional processors prompted the design of unorthodox architectures which could distribute processing by using parallel hardware configurations. The early trends were to use associative search hardware as subsystems within the central processor and the operations of these subsystems were closely associated with the operations of the central processor.[1] These systems, however, were restricted to small data bases because of their high cost. It is also evident that, with the present technology and the foreseeable future, it will not be feasible to build large scale pure associative memories as required by DBMS.

The desirable features sought in DBMS environment are:

(a) A large capacity and modular storage medium with low cost per bit,
(b) Ability to directly map logical data structures into physical data structures without using auxiliary structures,
(c) Variable length data formats,
(d) Fast retrieval and update of sets of data with respect to the requirements of on-line concurrent environments,
(e) Context (Boolean combinations of content) search operations assisted by total associativity,
(f) Complete instruction sets.

The solution to the cost limitations of associative memories in view of the desirable features of the DBMS environment was the introduction of the idea of using distributed logic in inexpensive large capacity circulating memory devices. Slotnick was the first to propose an associative file processor using a logic head per track device.[2] Healy and Parhami studied possible architectures which combined logic with a rotating bulk memory to achieve string and template matches in a context addressed manner.[3,4] Minsky proposed a scheme which involved an arrangement of the key items and data items grouped separately using the cylinder concept of a disk memory.[5] Parker partially designed a device that combined each head of a fixed head rotating memory with a single IC logic chip.[6] The studies mentioned thus far achieve only partial associativity and most of the fundamental DBMS operations had to be accomplished by the outside processor. Su, Copeland, and Lipovski were the first to study the design of a cellular processor on a rotating device to support the general data structures and requirements of DBMS.[7,8,9]

The overall design of the data structure, instruction set, and hardware architecture for RAP is presented. This design has been specified to the gate level. A discussion of cost and space estimates is presented in the conclusion. Details of this design can be found in a technical report.[10]

RAP ARCHITECTURE

Basic organization

An overall configuration of an operational RAP environment is given in Figure 1. RAP is an autonomous processor which communicates with an outside general purpose computer (GPC) only to receive its data base contents, to receive its compiled programs, and to send back the results of a user's requests.

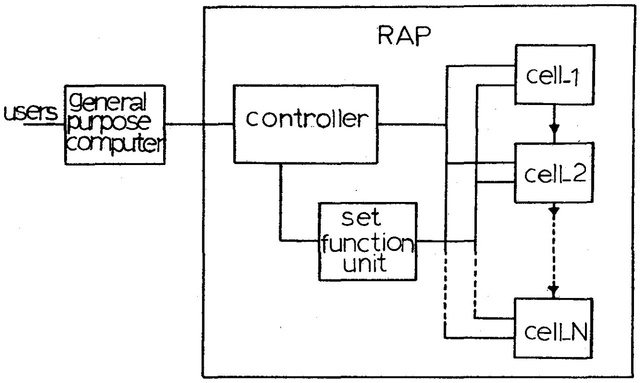


Figure 1—Overview of RAP architecture

The design is composed of a controller, an arithmetic set function unit, and a parallel organization of cells. A cell consists of a memory component and a logic component. The memory unit is one track of a rotating device such as a disk, drum, circular shift register, etc. The logic component is a microprocessor which acts as a "search machine" on data, directs data manipulation, and performs limited numeric computations required by data base processing. The set function unit is used to combine cell results to obtain a value computed over the total memory contents. The controller is responsible for overall coordination and sends control sequences to the cells, controls the set function unit, and executes decision commands and other RAP primitives that can be accomplished directly in itself.

The logical RAP data structure is stored directly so that no transformations are required. Data base contents are manipulated directly on their storage by their processors. The search criteria are transmitted to all of the cells simultaneously. It is evaluated as the memory contents "pass by" and the qualified contents are immediately manipulated or read out. Because the entire memory is processed for each instruction, inverted lists and other search aids would not enhance processing speed. More important, their absence necessarily eliminates the overhead of their maintenance. Since RAP has a parallel organization and its memory is associative, the response time is usually independent of data base size or content but does depend on query complexity. A query can be made up of one or more assembler instructions. Most instructions are executed within one rotation of the entire RAP memory contents.

Since data base management operates mostly in a concurrent environment, means must be provided to release the device as soon as possible from one operation to start another. It cannot tolerate excessive delays or overhead. The overall design philosophy is based on this principle. There are several hardware provisions throughout the design to achieve this principle.

Cell organization

Each cell consists of a rotating memory, a buffer, an information search and manipulation unit (ISMU), and an arithmetic logic unit (ALU). The basic logic blocks are displayed in Figure 2. Each cell lies on a serial communication path and receives status signals from its predecessor and sends signals to its successor. The cell also has connections for I/O and signals that are exchanged with the controller and set function unit.

Rotating memory

Data is read or written via fixed heads—one set for each cell—while the memory rotates under these heads. It takes one revolution of the store for its contents to be read from one end to the other. This time is called the latency. The associated memory with each cell is called a track. RAP memory space is the sum of the individual cell tracks.

A rotating bulk memory with high track capacity should be selected to achieve low cost per bit of storage. However, there is an upper limit on bit density since there is a processor associated with each cell. Equivalently, we require a lower limit on bit time—the time between two consecutively stored bits. It cannot be lower than a value determined by the speed of the processor circuits because logic and data/signal transmission must be completed in the time elapsed between two bits. A lower limit of 100 nanoseconds has been achieved as the bit time supported by the cell logic if implemented with the current IC technology.

Buffer

As the memory rotates under the heads, it is read, circulated through logic, and written back after a time delay. This delay is proportional to the length of a shift register buffer placed between read and write heads. This buffer has been designed to have a length of 1024 bits which holds a sufficient amount of data exposed to the cell logic to support the logical data structure.

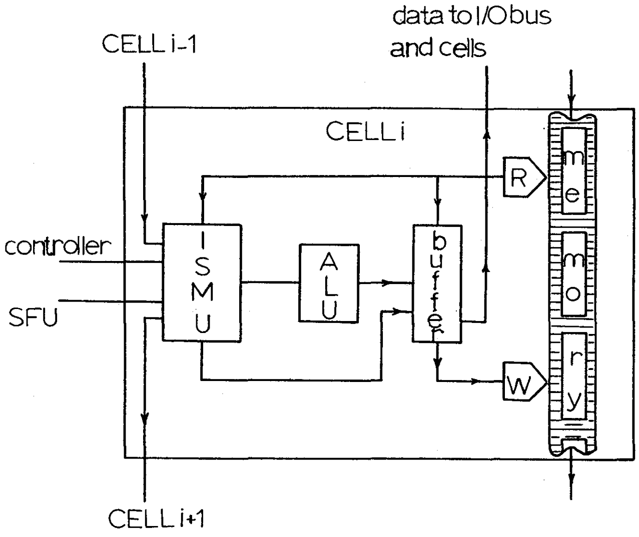


Figure 2—Overview of cell architecture

Data structure

A general data structure called RAP relations (similar to normalized relations as defined by Codd[11]) are stored with respect to a fixed track format. Details of the data structure and track format are given in the following sections. It suffices to say that a relation can be viewed as a "table" of data whose rows will make up the blocks of data stored on a track. Rows are called tuples because they represent a p-tuple of values. Both the ISMU and ALU circuits are designed to function on variable length data fields within the data blocks.

Contents of the memory are composed of blocks of data and/or garbage. The functions of storage allocation and garbage collection are accomplished directly in the hardware. Each cell uses the buffer to pack the data on the track toward the beginning of the track by "short circuiting" unwanted items. This accumulates the garbage at the end of the data. New data is first inserted at the garbage space of related tracks and, if more space is required, a new track is initialized and used to store the rest of the data. Garbage collection takes place in parallel with other operations without the need of user control or intervention.

Information search and manipulation unit

This unit is responsible for inter-cell communication, decoding of the commands sent from the controller, evaluation of data search criteria, I/O data transfers, and control of the ALU for data modifications.

Arithmetic logic unit

This unit contains a serial adder, multiplier, control counters, and logic for arithmetic computations and modifications. Logic for intermediate set function calculations (e.g., summation, maximum, etc.) are also present.

Concurrency facility

The RAP concurrency facility employs a foreground-background principle implemented in hardware so that long programs can be run on preemptive basis.

Set function unit

The set function unit (SFU) provides the logic to calculate the set functions count, sum, maximum, minimum, and average over the entire cell memory. Intermediate results are stored in individual ALU registers of the cells. Arithmetic is accomplished through a serial adder and a serial divider which work on the intermediate results as they are collected from the cells. Also provided are counters for control sequencing as well as data gathering from the cell units and for transmitting the SFU result to the controller.

Controller

The controller is responsible for the overall coordination of the cell processors. It loads the search criteria units of ISMUs, energizes opcode and mode lines, senses the end of an operation, and repeats the cycle for the next primitive. The micro-orders cause stack buffer contents associated with instructions to be popped off and bussed onto cell lines during data block gap intervals. The controller also executes certain primitives that can be accomplished directly in itself by making use of the contents of registers whose values have been stored from previous operations.

Other operations, such as data transfers to and from the GPC, parity bit generation, serial-parallel conversion, and error checking procedures, which are common routines that exist in present devices, will be incorporated into the final design.

Role of the general purpose computer

A GPC to be chosen for interfacing with the RAP device should provide the following main functions:

(a) Support a data communication environment for users with proper I/O facilities,
(b) Compile user queries expressed in a high level query language into RAP primitives,
(c) Transfer compiled RAP primitives to the RAP controller with associated data used as operands in the format required by the RAP controller,
(d) Transfer data to cell storage for data base creation and expansion,
(e) Support a concurrent processing environment and thereby administer scheduling of RAP program entry queues,
(f) Control data base security and integrity,
(g) Maintain relation names, domain names, and data value encoding tables.

RAP DATA STRUCTURE

The data structure chosen for RAP is a modified version of the relational model introduced by Codd.[11] This model has been proven to be general enough to build other known useful data structures and, more important, present the same information conveyed as other data structures in a way that achieves a high degree of data independence.

RAP relational structure

We can view a normalized relation as a table of data about a set of similar entities. The table heading is the relation name, the column headings are the entity's attributes which are called the domain names, and a row represents a p-tuple, or simply a tuple, of values—one for each domain—which describes an entity. A relation contains a varying number of tuples and a relational data base contains several interrelated relations through common domains.

A candidate key of a relation is a minimal combination of domains whose elements (values) uniquely identify every tuple in that relation. A primary key is one of the candidate keys with which the unique tuple identification is based for implementation.

A RAP relation is a normalized relation of the type described above except that duplicate tuples are allowed. One restriction is that the degree cannot be higher than a number pmax depending on the size of the values of a domain. This limitation is imposed by hardware considerations. It would suffice to state here that due to variable word length representation of RAP domains, the range for a RAP relation's maximum degree is 29 ≤ pmax ≤ 101. We expect this to be sufficiently high enough for most applications. If this is not the case, then two or more relations can be made from the larger one. This is accomplished by partitioning the domains and repeating the primary key with each subrelation.

Track format

Since RAP eliminates intermediate mapping structures, a method of representing the data structure directly on the storage had to be found. Due to the linear nature of a memory track, a direct mapping of data structure would require it to be linearized. The relational structure lends itself to linear form easily as shown in Figure 3. The tuples of the relation, which are themselves linear representations of domain values, can be stored one after the other on a track. This structure is similar to a file on a conventional disk. The first two data blocks contain relation and domain names respectively and act as "header" blocks. Each succeeding block contains the concatenated value items in a tuple. The order of domain names determines the order of the values in each tuple. If a relation has too many tuples to be stored on one track, then several cell tracks are used. The hardware requires the relation and domain names to be repeated once on each track of a relation and that no cell can have tuples from more than one relation.

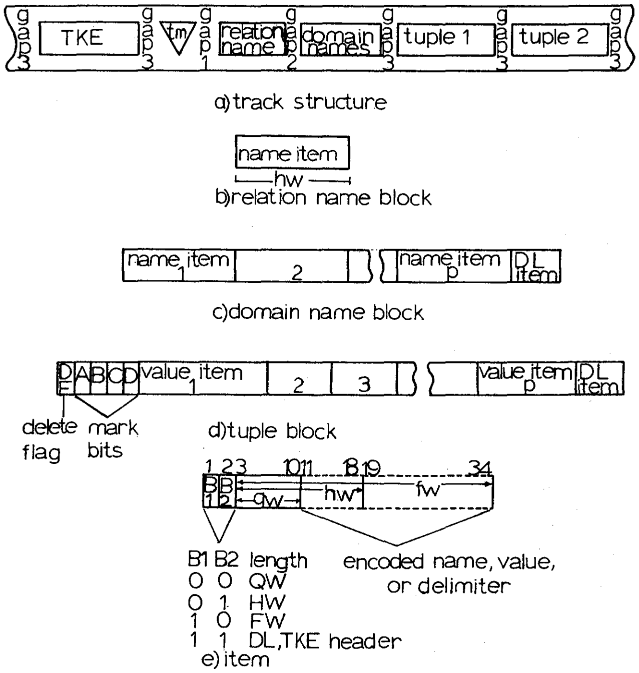


Figure 3—Track format

All names, values, and delimiters are items of encoded bit patterns. The relation name block is made up of one fixed length item. Domain and tuple blocks can have a variable number of items in different relations but not within the same relation. The end of these blocks is indicated by a delimiter item (DL). Blocks are separated by fixed length interrecord gaps. The beginning of a track is indicated by a marker which is detected electronically. This marker also implies the physical end of a track. The logical end of a track is indicated by a tuple block which carries a delimiting "track end" (TKE) item stored in the first value item's position. Items that make up domain and value blocks can also be variable length. However, these lengths must either be 32, 16, or 8 bit encodings. Each item is preceded by a two bit code to indicate the item's length. The fourth code is used to specify DL or TKE items which are distinguished from each other by the following bit. There is an upper bound on the length of a RAP relation tuple which is determined by the length of the cell buffer (1024 bits).

Tracks for a relation are allocated one track at a time as needed. Relation tracks are not required to follow sequential or contiguous track addresses since each relation track is identified separately. This results in different relation tracks intermixed with each other. By making each track a separate entity by itself and spreading the relations across noncontiguous cells, output efficiency is greatly increased. This is because several contiguous tracks are serviced as one channel group. Output occurs from one qualified cell in each channel group and outputs from these groups are interleaved to achieve a piped transmission.

There are five control bit positions at the beginning of each tuple. The first bit is the delete flag (DF). If this bit is on, it indicates that tuple is deleted and garbage collection hardware can "erase" the tuple. Garbage collection on each track is handled dynamically. The data is packed toward the beginning of a track. In case of insertions, all of the relation tracks are examined and their available garbage tuples are filled with incoming tuples. If still more insertions are to be made, a new cell and its relation track can be allocated automatically.

The A, B, C, and D are mark positions composed of one bit each. If a tuple is T-marked, T being any combination of these 4 bits, the corresponding mark bits are turned on. Likewise, a tuple is said to be T-unmarked if the T combination of bits are turned off. There are several instructions for marking tuples and/or using the markings as extra data values for qualifying sets of tuples. These bits allow the results of one instruction to be used by another. This greatly extends the associative capability of RAP.

A fixed length gap is required between every two blocks. The lengths of these gaps are proportional to the amount and speed of logic required between block operations.

Preformatting of a track involves writing of all the control information of the block types on a track and separating the blocks by gaps. Relation and domain blocks must be filled in by their names. Tuple blocks are written until the physical end of the track. Their DL identifiers and the length codes in the first two bit positions of each item must also be filled in. The data portion of value items does not contain any information at that time. The logical track end is indicated by writing the identifier TKE in the first value item position of the very first tuple block. The first value item position of each tuple block starts at the sixth bit position leaving room for the delete flag and mark bit positions. Care is taken to make all of these five bit positions zero while preformatting each tuple block.

RAP INSTRUCTION SET

Basic relational operations

The RAP instruction set is designed to construct the basic operations necessary to support relational data bases and, at the same time, to be feasible for hardware implementation.

These basic operations are:

(a) Selection,
(b) Implicit join,
(c) Set operations,
(d) Projection,
(e) Free variables,
(f) Arithmetic set functions,
(g) Simple arithmetic update.

Selection is performed by applying a Boolean search predicate to each tuple of a relation and "marking" or reading the tuples satisfying the predicate. The implicit join operation allows values retrieved from one relation to be used as the retrieval criterion on another relation. The association is made through domains common to both relations. The set operations of union, intersection, complement, and difference can be done on the selected subsets of a relation. Projection is the act of selecting a subset of domains to be retrieved and eliminating duplicate values after a possible selection has occurred. A "free variable" is the term given to an implementation of the following capability.[12] It involves the selection of tuples based on the values of domains which occur in other tuples of the same relation. RAP programs which accomplish projection and free variables require explicit iteration.

Structure and operation of the instructions

Many RAP primitives reflect in their structure the basic characteristics of a DBMS query. These primitives specify an *operation* to be performed, a data *specification* section indicating what data is to be operated upon, and a *qualification* section specifying what conditions must be met before the operation can be fulfilled. The general format of these primitives is:

⟨LABEL⟩⟨OPCODE⟩[⟨SPECIFICATION⟩:
⟨QUALIFICATION⟩]

The label is an optional symbolic address of an instruction. The opcode specifies the operation. A specification has the format:

RN(DN1, DN2, . . ., DNK)

where RN is a relation name and DN1, DN2, . . ., DNK is an optional domain list. There is a hardware limit k on the number of K domains that can be included for any specification. A qualification is a Boolean expression of conditions on the values of the domains of a relation and on the mark bit values of the tuples in which the domain values are stored. It can take one of the following forms:

(a) Null (a qualification which does not specify any condition), every tuple of the relation is considered to satisfy the qualification,
(b) Q1^Q2^· · ·^QK, denoting conjunction,
(c) Q1∨Q2∨· · ·∨QK, denoting disjunction.

where K must be less than or equal to k—the hardware limit of the number of parallel comparators—and Qi is one of the following:

(1) ⟨RN⟩·⟨DN⟩ ⟨COMPARATOR⟩ ⟨OPERAND⟩,
—where DN is a domain name in relation RN,
—COMPARATOR is one of =, ≠, <, ≤, >, ≥,
—OPERAND is one of:

- ⟨REG⟩ (a register),
- integer,
- literal (characters bounded by quotation marks),
- a GPC program variable name (enclosed in parentheses),

(2) RN.MKED(T),
(3) RN.UNMKED(T),
where T is replaced by one of the following mark bit combinations:

A, B, C, D, AB, AC, AD, BC, BD, CD, ABC, ABD, ACD, BCD, ABCD.

Permutations are considered to be identical,
(4) CELL (I), where I is the integer identification of a cell.

Consider the following example. Assume a ternary RAP relation EMPLOYEEwith domains NAME, NCHILDREN, and SALARY and further assume that one of its tuples is stored in track format with the values shown below.

(a) DF=0
A=1
B=0
C=1
D=0
(b) NAME=CLARK
(c) NCHILDREN=3
(d) SALARY=9500

Some of the qualifications that this specific tuple satisfies are:

(a) NULL
(b) EMPLOYEE.MKED(A)
(c) EMPLOYEE.UNMKED(BD)
(d) (EMPLOYEE.NAME='CLARK') $\wedge$
(EMPLOYEE.MKED (AC))
(e) (EMPLOYEE.NCHILDREN $\neq$ 5) $\vee$
(EMPLOYEE. SALARY > 5000) $\vee$
(EMPLOYEE.MKED(A))

At this stage, even without seeing the details of the instructions, it is possible to show how set operations can be implemented by using marking qualification logic. Assume that at one instance tuples of a relation are A and B-marked by criteria-1 and criteria-2 respectively. We denote the subset of tuples A-marked by criteria-1 by "SETA" and those B-marked by criteria-2 by "SETB". The following qualification statements can be written for the set operations:

(a) SETA$\cap$SETB (intersection) qualification:
RN.MKED(AB)
(b) SETA$\cup$SETB (union) qualification:
(RN.MKED(A))$\vee$(RN.MKED(B))
(c) SETA-SETB (difference) qualification:
(RN.MKED(A))$\wedge$(RN.UNMKED(B))
(d) $\neg$ SETA (complement) qualification:
RN.UNMKED(A)

Each command is formulated as an assembler language instruction for the RAP machine. One assembler instruction is implemented as one controller micro-code sequence which activates the hardwired logic in the cell circuits. The opcodes for each instruction will be given followed by a brief explanation. Instruction timings are given in the following section.

Retrieval commands

These commands are used in the process of locating the qualified rows of relations, and, if required, outputting them to the GPC. The commands included in this group are:

(a) MARK(⟨T⟩): Indicates the tuples of a relation which satisfy a Boolean condition on its values by setting "marking" bits,
(b) RESET(⟨T⟩): Resets marks,
(c) READ: Transfers fields of data from qualified and/or marked tuples,
(d) READ_REG: Transfers the contents of registers from the controller,
(e) CROSS_MARK(⟨T⟩): Marks a second relation based on the values of a previously marked relation (i.e., a hardware implementation of the implicit join operation),
(f) CRS_COND_MARK(⟨T⟩[⟨T^1⟩]):
A variation of CROSS_MARK; used for cross marking several relations into a single relation,
(g) GET_FIRST_MARK(⟨T⟩): Used for queries which involve the selection of tuples based on values which occur in other tuples of the same relation (e.g., used within a program implementation of the free variable and projection operations). It finds the first marked tuple of a relation, resets it, and takes the value of a domain from that tuple and marks other tuples in the same relation whose same or other compatible domain satisfy a comparison on that value. It can also store value items of a tuple into the controller registers to be used as arguments in subsequent operations.
(h) GET_FIRST: Places the specified values of the first marked tuple found into controller registers and resets the mark,
(i) SAVE: Stores values from a single tuple into controller registers.

Update commands

These commands provide value replacement by direct insertion or after arithmetic operations. The updated values are written back immediately so that a separate rewrite command is not required. The commands in this group are:

(a) REPLACE(⟨T⟩): Replaces the value of a domain of a set of qualified tuples with a new value,
(b) ADD(⟨T⟩): Adds a constant, contents of a controller register, or contents of another numeric domain to a domain of a set of qualified tuples,
(c) SUB(⟨T⟩): etc.,
(d) MUL(⟨T⟩): etc.,
(e) DIV(⟨T⟩): etc..

The optional marking capability of these instructions is used for recovery from device or software crashes. As each update takes place, the combinations of T-mark bits are set. If a crash occurs, the operations can be resumed on tuples with T-markings still off.

Set function commands

These commands compute functions over qualified sets of data. Similar to update commands the memory contents are evaluated "in place" so that transportation of large amounts of data is eliminated for simple computations. The commands included in this group are:

(a) SUM: Sums the values of a set of qualified tuples,
(b) COUNT: etc.,
(c) AVERAGE: etc.,
(d) MAX: etc.,
(e) MIN: etc..

Insertion and deletion commands

The commands are:

(a) INSERT: Inserts a new tuple into a relation,
(b) DELETE: Deletes qualified tuples of a relation,
(c) DROP_DOMAIN: Deletes a domain from a relation.

Data base creation and destruction commands

The commands are:

(a) CREATE: Preformats a cell so that it may be used to store and manipulate tuples of a relation,
(b) DESTROY: Removes all formats and data from a single cell or all cells containing data from a specified relation.

Decision and transfer commands

As in any programming language, certain instructions are required for testing system indicators, providing conditional and unconditional transfers within a program, and indicating termination. The following instructions provide these functions:

(a) TEST: Sets the contents of a status register based on the contents of markings,
(b) BC: Transfers control to another instruction if a condition is met,
(c) EOQ: Signals the end of a query.

SUMMARY OF INSTRUCTION TIMINGS

Table I gives the time required for the execution of each of the RAP instructions.

Table I—Instruction Timing

TYPE OF OPERATION	INSTRUCTION	EXECUTION TIME IN NUMBER OF REVOLUTIONS UNLESS OTHERWISE SPECIFIED
Retrieval commands	MARK	1
	RESET	1
	READ	Minimum=1/2 on average, Maximum=Number of qualified tracks occupied by the relation
	READTREG	Negligible
	CROSS_MARK	Depends on data base and k (Reference 10)
	CRS_COND_MARK	Same as CROSS_MARK +1
	GETTFIRST_MARK	1 +Fraction
	GETTFIRST	1
	SAVE	1
Update commands	ADD	1
	SUB	1
	MUL	1
	DIV	1
	REPLACE	1
Set function commands	COUNT	1 +Fraction
	MAX	1 +Fraction
	MIN	1 +Fraction
	SUM	1 +Fraction
	AVERAGE	2 +Fraction
Insertion and deletion commands	DELETE	1
	INSERT	Minimum = 1, Maximum = 2
	DROPTDOMAIN	Maximum number of relation tuples per track
Data base creation and destruction commands	DESTROY	Fraction
	CREATE	1
Decision and transfer commands	TEST	Negligible
	BC	Negligible
	EOQ	Negligible

SELECTED SAMPLE QUERIES

Some sample queries will be programmed with the RAP instruction set. The samples are similar to those described in the literature for languages SQUARE,[12] SEQUEL,[13] and ALPHA[14] which were proposed as high level query languages for relational data bases. These languages have been proven to be complete in the sense that they are at least as powerful with respect to query formulation as the relational calculus. The RAP instruction set includes all the capabilities of these languages. All of the queries used as examples in the above languages have been programmed in RAP.

The data base used for the following examples consists of the following relations:

EMP(NAME, DEPT, MGR, SAL)
LOC(DEPT, FLOOR)
SALES(DEPT, ITEM, VOL)

The employee relation has a tuple for every employee giving his name, department, manager's name, and salary.

The location relation LOC gives the floor on which each department is located, that is, it has a tuple for each department floor pair. The SALES relation has a tuple indicating the yearly volume sold for an item in each department.

Q.1 Find the employee whose salary is greater than that of any employee in the SHOE department.

The RAP program is:

(a) MAX [EMP(SAL):EMP.DEPT = 'SHOE'],
(b) MARK(A) [EMP:EMP.SAL 〉 (REGF_1)],
(c) EOQ

Q.2 List the names and managers of employees in the SHOE department with salaries greater than 10,000.

The RAP program is:

(a) READ [EMP(NAME,MGR):(EMP.DEPT = 'SHOE') (EMP.SAL 〉 10000)] [WORKAREA],
(b) EOQ

Q.3 Move the location of the TOY department to the second floor.

The RAP program is:

(a) REPLACE [LOC(FLOOR):LOC.DEPT = 'TOY'] [2],
(b) EOQ

Q.4 Find the items sold by departments on the second floor.

The RAP program is:

(a) MARK(A) [LOC:LOC.FLOOR = 2] Mark departments on floor 2,
(b) CROSS_MARK(C)
[SALES:SALES.DEPT = LOC.DEPT:
LOC.MKED(A)]
Cross mark into SALES tuples the matching departments in the marked LOC tuples
(c) READ [SALES(ITEM):SALES.MKED(C)]
[WORK-AREA]
Read items sold by departments on the second floor
or
(b^1) CROSS_MARK(BC [SALES:SALES.DEPT
=LOC.DEPT: LOC.MKED(A)]
(c^1) (1) L1 GET_FIRST_MARK(A) [SALES:SALES.
ITEM= SALES.ITEM:SALES.MKED(b)]
(2) RESET(ABC) [SALES:SALES.MKED
(ABC)]
(3) TEST B-RAIL
(4) BC L1,RAILTSTAT(B)
(1) through (4) is a projection
operation on the answer found in b^1
(5) READ [SALES(ITEM):SALES.MKED(C)]
[WORKAREA]
read projected results that are
C-marked
(d) RESET(C) [SALES]
(e) EOQ

TABLE II—Cost and space estimates for a single cell

	CHIP COUNT/TOTAL COST (INCLUDING MULTIWIRE ™WIRING AND CIRCUIT BOARDS)		
Device complexity	k=1	k=5	k=5 & Concurrency
Catalogued SSI+MSI components	300/$425	399/$575	458/$657
All MSI (some to be made specially)	70/E$156	101/E$224	114/E$253
MSI+LSI (some MSI and all LSI to be made specially)	28/E$124	40/E$180	42/E$190

COST AND SPACE ESTIMATES

A preliminary logic design of the cell hardware has been completed to the gate level. This design has provided estimates for the cost and space requirements of the RAP hardware. The number of parallel comparator units k must be considered in terms of software implications and hardware costs. At this stage an intuitive value of 5 appears to satisfy a wide range of software applications. Table-2 presents the space and cost figures. E$ denotes estimated costs.

The relationship between the k values is linear. The R/W buffer is an existing LSI component. All components other than the LSI are constructed from T^2L some of which are Schottky and high speed types. The power dissipation for an implementation at the MSI scale would be 17, 25, and 29 watts per cell for k=1, k=5, and k=5 & concurrency respectively and a lower dissipation will result with LSI implementation.

The costs in the table are given for the full capabilities of the system, that is, all of the instructions are included. The first row gives the catalogued small quantity figures which can be used for the construction of an initial prototype. The controller circuitry dealing with the RAP functions is about the size of one cell. This excludes the read only and other buffer and stack memories as well as the circuits in the controller I/O function unit.

A review of current industrial technology indicates that disk memories could contain a track capacity of $0.5*10^6$ bits with a latency less than 50 milliseconds. Two hundred cells with this track capacity can accommodate a large data base on the order of 10^8 bits of compressed data. For very large data bases RAP can be used as an intelligent virtual memory to be backed up by conventional large bulk memories.

ACKNOWLEDGMENT

This research was supported in part by the National Research Council of Canada.

REFERENCES

1. Dugan, J. A., R. J. Green, J. Minker, "A Study of the Utility of Associative Memory Processors," *Proceedings of the ACM National Conference*, 1966.

2. Slotnick, D. L., "Logic Per Track Devices," *Advances in Computers*, Academic press, 1970.
3. Healy, L. D., K. L. Doty, G. J. Lipovski, "The Architecture of a Content Addressed Segment Sequential Storage," *Proceedings of FJCC*, 1972.
4. Parhami, B., "A Highly Parallel Computer System for Information Retrieval," *Proceedings of FJCC*, 1972.
5. Minsky, N., "Rotating Storage Devices as Partially Associative Memories," *Proceedings of the ACM SIGFIDET Workshop on Data Description*, Access, and Control, 1972.
6. Parker, J. L., "A Logic Per Track Retrieval System," *IFIP Congress*, 1971.
7. Su, S. Y. W., G. P. Copeland, G. J. Lipovski, "Retrieval Operations and Data Representations in a Context Addressed Disk System," *Proceedings of ACM Programming Languages and Information Retrieval Interface Meeting*, 1973.
8. Copeland, G. P., G. J. Lipovski, S. Y. W. Su, "The Architecture of CASSM: A Cellular System for Non-numeric Processing," *First Annual Symposium on Computer Architecture*, 1973.
9. Copeland, G. P., S. Y. W. Su, "A High Level Data Sublanguage for Context Addressed Segment Sequential Memory," *Proceedings of the ACM SIGFIDET Workshop on Data Description, Access, and Control*, 1974.
10. Ozkarahan, E. A., S. A. Schuster, K. C. Smith, *A Data Base Processor*, Computer Systems Research Group TR-43, University of Toronto, September 1974.
11. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *CACM*, 13, 6, 1970.
12. Boyce, R. F., D. D. Chamberlin, W. F. King III, M. M. Hammer, *Specifying Queries as Relational Expressions: SQUARE*, IBM Technical Report RJ 1291, IBM Research Laboratory, San Jose, California, October 1973.
13. Chamberlin, D. D., R. F. Boyce, "SEQUEL: A Structured English Query Language," *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, May 1974.
14. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, 1971.

The datacomputer—A network data utility*

by THOMAS MARILL and DALE STERN
Computer Corporation of America
Cambridge, Massachusetts

OVERVIEW

The Datacomputer is a large-scale data management and storage utility for use by a network of computers. The system is designed to provide facilities for data sharing among dissimilar machines, rapid access to large on-line files, storage economy through shared use of a trillion-bit store, and improved access control.

The present paper provides a conceptual overview of the system. Detailed treatment of the access language, software architecture, and relation to other developments in the database field[4,5,7,8,9] will be taken up in subsequent papers.

NETWORKS AND UTILITIES

Starting in the early 1960s, the idea that stand-alone computers could cooperate through communication facilities began to be explored,[1] and the concept of the resource-sharing network evolved.[2] In such a network, each computer draws on the others to supplement its own resources of hardware, software, and data. Today, the best-known network of this type is the Arpanet,[3] which ties together some forty-odd computers of different types.

Within a resource-sharing network, there is a natural tendency toward specialization of network nodes. Thus, for example, medium-scale machines with good time-sharing facilities will be used for interactive processes, but heavy scientific computation will tend to be passed to other machines that are particularly adept at such tasks. The factoring of problems into their constituents, the assignment of these constituents to the appropriate machines, and the recombination of results will tend to become an automatic process.

In the limit, specialized network nodes become what may be termed "utilities", that is, machines which perform a restricted range of functions solely for the benefit of the other machines. The Datacomputer is a network utility in this sense. It is entirely specialized for the performance of data management and storage functions. It offers resources to other machines on the net but does not draw on the resources of these machines.

One may speculate that the trend toward specialized network utilities will continue, and that the traditional stand-alone general-purpose machine will eventually disappear from the scene. The computer world envisioned in such a speculation might consist of a network containing a few very large Datacomputer-like systems, a few very large computational utilities ("number crunchers"), and a large number of small human-interaction units (such as intelligent terminals), having limited computational power and local storage. It is not clear that anything else is needed.

The justification of network utilities must primarily, of course, be made on economic grounds, by demonstrating that economies of scale and economies of specialization can be realized. In the case, specifically, of a data utility, there is an added justification: centralization reduces the severity of the technical problems of data sharing and may also alleviate some of the problems associated with privacy. If all data is kept in one box, one knows where to go look for it; by the same token, one knows where the control and protection procedures must be applied.

DESIGN CONCEPTS

Logically, the Datacomputer system can be viewed as a box which is shared by a variety of external processors, and which is accessed in a standard notation called "datalanguage." (See Figure 1.) The present section discusses the principal concepts underlying the design of the system.

Network data sharing

The Datacomputer provides data sharing services within a network environment. There are three principal design implications of this fact.

Data conversion

A database stored on the Datacomputer is sharable by all computers having access to the system. Thus, a single database is shared not only among users of different interests, but among users of different hardware. Character codes, floating point number representations, and word sizes vary from user to user; so do the representations of variable length and variable structure, as well as high level data structure attributes. The

* Work supported by the Advanced Research Projects Agency, Department of Defense.

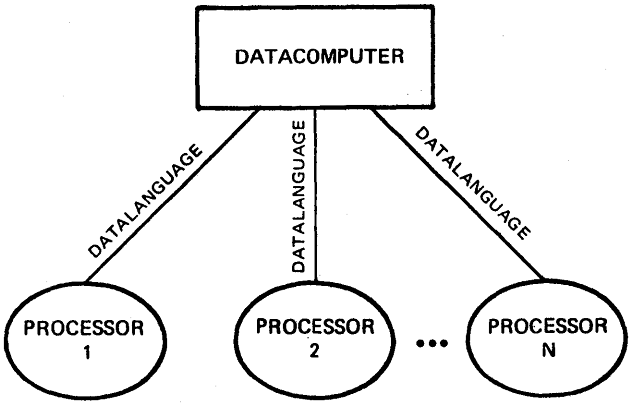


Figure 1—Logical view of datacomputer

Datacomputer system is required to perform translations between various hardware representations and data structuring concepts.

Characters, bytes, and numbers are stored under the control of the machine storing the data. The machine reading the data specifies the format it requires. As data is output, the indicated data conversions are performed.

Self-contained requests

In most approaches to data management (for example, the CODASYL approach[4]) the assumption is made that the data management system is in close contact with the application program. Thus the data management system can rely on the full capabilities of an application language (for example, COBOL) as being immediately available for processing the data.

This is not the case in a network environment, where the bandwidth between the application program and the data management system is relatively low. Thus, datalanguage must be designed to allow self-contained requests to be shipped to the Datacomputer to be executed there *in toto.*

Consider, for example, the problem of updating a large personnel file to reflect an across-the-board salary increase of 5 percent. In a conventional approach, the application program would sequentially obtain every record by making appropriate calls to the data management system, update the salary field, and replace the record (or build a new file) by calls to the data management system.

In a network, such an approach would be undesirable for large files, since it would require the entire file to be shipped twice, once to the application program, and once again back into storage. Accordingly, datalanguage is designed so that self-contained requests may be shipped to the Datacomputer from the application program. The Datacomputer itself performs the indicated function and signals the application program that the job has been completed, without requiring the records to be shipped to the application program.

Datalanguage does not, however, prevent the user program from generating a request which would cause the Datacomputer to ship an entire file to the requesting computer. That is, the Datacomputer can be used as a "file manager" in the style of the TABLON system,[5] as well as a data management system. For small files, this may be the preferred mode of use. For example, a short document that needs to be edited might best be shipped as a unit to the machine on which the editing will be performed, and then shipped back for storage.

Computer-oriented

The Datacomputer communicates with programs that run on remote machines. The fact of remoteness precludes the use of simple subroutine calls or similar means of communication conventionally used within a single machine. The communication, furthermore, is not with people at terminals, who can be expected to make intelligent responses when failures or unusual circumstances occur, but with programs. Hence, all synchronization messages, error messages, language statements, and file descriptions must be creatable and readable by programs; likewise, a facility for checkpointing by user programs is required.

Large on-line files

The Datacomputer is designed to have an on-line storage capacity of a trillion bits and to accommodate a wide variety of file sizes. In particular, the system handles files whose size approaches the total available space, that is, files in the trillion-bit range. To achieve efficient access to such files, two special facilities are included.

Inverted file structure

No adequate large file system can be designed without providing some mechanism for calculating the location of data in storage, given the attributes of the data to be retrieved. In the Datacomputer, this capability is achieved through a system of inverted files.*

At the user's option, files stored at the Datacomputer are totally or partially inverted. Once the file has been loaded, the inversion tables are maintained automatically by the system and need not be of concern to the user. Requests against a file may be composed without knowledge of the inversion options that have been selected for that file. The system will use the inversions, to the extent that they apply in a particular request, to limit the amount of sequential search that must be performed, thereby speeding up its retrieval process.

Multiple staging strategies

Internally to the Datacomputer, all data is physically organized into pages which move among the three levels of

* In a *direct file* one lists, for each entity, the properties of that entity. In an *inverted file* (also called *inversion*) one lists, for each property, the entities (or the location of the entities) having that property.

storage: primary (core), secondary (disk), and tertiary (mass store). The movement of pages is dictated by various staging strategies. The particular strategy used is selected by the system to optimize the requests currently being executed. The fact that the Datacomputer can itself select among the available strategies hinges on the fact that entire requests are transmitted to the system, informing the system at one time of the user's intent with respect to a given file.

Examples of staging strategies are as follows:

(i) Move the whole file to disk and work from disk. This strategy is applicable to small files that easily fit into the available secondary storage buffer area.

(ii) Move pages from tertiary store to core, process the pages, and output directly from core, bypassing disk. This strategy is applicable, for example, in the case where only a small portion of the data read from tertiary storage is to be sent to the user.

(iii) Break the request down so as to operate on segments of a file, and stage to disk one segment at a time. This strategy becomes particularly effective when information is available (from the inversion tables, for example) to indicate that some segments are not needed to fulfill the request, and can therefore be skipped.

Access regulation

The problem of controlling the access of programs to data in a general-purpose machine is notoriously difficult. By definition, a general-purpose environment allows the programs within it enormous latitude in the functions they can perform, and it appears that programs can often be written to circumvent existing access regulation procedures by taking advantage of coding errors in the operating system, hardware bugs, momentary malfunctions, or operational errors that arise in unexpected circumstances. Such hostile programs are sometimes able, without authority, to access data, delete data, or crash the system and prevent other users from legitimately accessing data.

In the environment of the Datacomputer, the situation is quite different, since the system is logically a closed, dedicated, special-purpose box, which responds only to a limited set of commands and does not provide a general-purpose computing facility. A hostile user program cannot be run on the box because the box does not run user programs. The approach can inherently provide stronger guarantees that programs without proper access authority will not be able to access or damage data contained in the Datacomputer. It is possible—though this needs to be explored further—that the Datacomputer approach lends itself to a proof that unauthorized access cannot occur.

Economy of scale and specialization

A variety of mass storage devices are coming on the market. These devices—the Ampex TBM, IBM 3850, Precision Instrument 190, among others—all have very high price tags, ranging from several hundred thousand to several million dollars, depending on configuration. They all, however, provide very low per-bit unit cost, with the lowest per-bit cost occurring in the largest configurations. Thus, while few stand-alone installations could afford the entry price, by pooling many users' requirements into a shared Datacomputer facility, the low per-bit cost of the mass store can be passed on to the users.

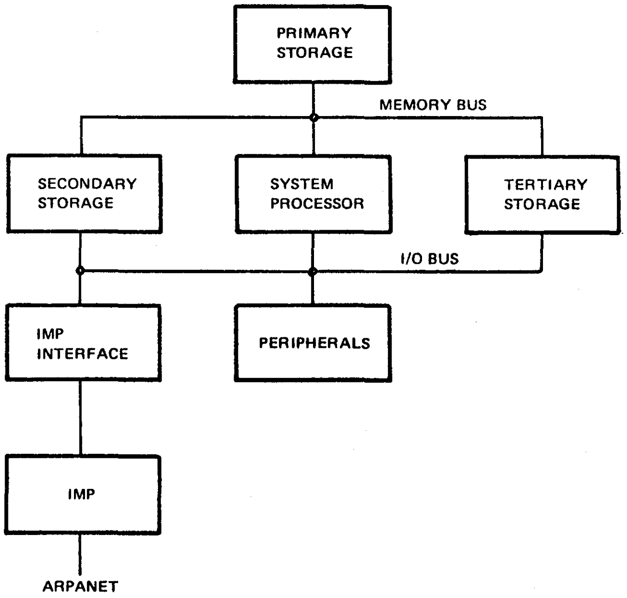


Figure 2—Hardware overview of system

The savings can be substantial. Disk storage equipment (at the low end of the currently-available price-range) costs about $20 per megabit of storage. Mass stores cost about $1 per megabit, some twenty times less. All of these prices may be expected to decrease as technology improves, but there is no reason to suppose that the relative advantage of the economy of scale will not remain.

Certain additional economies can also be realized through specialization. In designing a specialized system it is possible to choose hardware and implement software in such a way as to optimize for the particular application, since there is no requirement to provide general-purpose services. In the particular case of the Datacomputer, it is possible to take advantage of new technologies as they become available, by making internal modifications and additions to the hardware and software of the system. This can always be done so long as datalanguage remains invariant, since the user program does not "see" the hardware or software of the system.

HARDWARE OVERVIEW

The architecture of the system is shown in Figure 2. The system processor is a DEC System-10 (PDP-10). Memory is present at three levels: core, disk, and TBM.[6] Peripherals are used for software development and for input

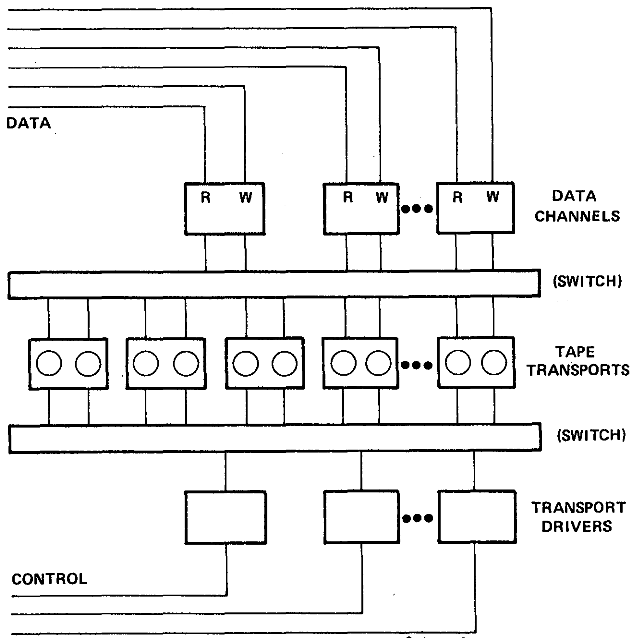


Figure 3—Ampex TBM configuration

of data from tape. The system is interfaced to the Arpanet IMP,[3] which in turn interfaces to two 50 kilobit/second telephone lines into the network.

Figure 3 shows in greater detail the configuration of the Ampex TBM tertiary storage subsystem. The system consists of three types of components, interconnected by two banks of switches. Channels are subdivided into two half-channels, one for reading and one for writing, each with a 6 megabit/second bandwidth. Each tape transport has two tapes, with a combined capacity of about 10^{11} bits; the maximum configuration has 64 tape transports. The transport drivers, or controllers, are switchable to any of the transports. In operation, a transport driver is switched to a tape transport, which is in turn switched to a data channel. Control information is passed to the transport driver, and data flows through the data channel. Data is written redundantly on the tape in a helical video scan with a density of 1 megabit/square inch. The average access time is 15 seconds.

DATALANGUAGE

Datalanguage is the language in which all requests to the Datacomputer are stated. Datalanguage includes facilities for data description, for database creation and maintenance, for selective retrieval of data, and for access to a variety of auxiliary facilities and services.

Datalanguage is a high-level language, which presents the user with a view of data which is independent of considerations of the physical devices on which the data is stored. The end user need not concern himself with search and scheduling techniques that are device-dependent.

Data representations are a special concern, because of the diversity of the user community. Data attributes which are ignored in other systems must be specified in this environment. The user must be able to map the data representations and data structuring concepts of his own machine onto those of the Datacomputer.

A basic characteristic of datalanguage is that all data is described. Descriptions are stored in the Datacomputer directory and are available to the user program in machine-readable format. A description contains the information needed to interpret the data, that is, information on data representations and structure.

An I/O transaction requires two descriptions: one for the data as it is stored—the "file description"—and one for the data as it comes in or goes out over the network—the "port description." Through the file description, the data administrator has control of how his data will be formatted on the Datacomputer. He can choose the representation that corresponds to the way the data will be accessed most frequently. In this way, the computation needed for reformatting is minimized, and higher bandwidths in and out can be achieved. Through the port description, the end user controls how the data as he sees it on his machine is formatted.

The data description facilities for ports and files are identical. In moving data between a file and a port, the Datacomputer performs the necessary reconfigurations of the data, including conversion from one elementary data type to another and pruning and reordering of branches in a hierarchical data structure.

Figures 4 and 5 show a port and file description, respectively, for a file of weather data. The port, called RESULTLIST, contains a list of "structs", called RESULT. Each RESULT has a city, date, and a minimum and maximum temperature. In this particular example, all of the data elements are fixed-length ASCII strings.

The file, called WEATHER, is tree-structured. Each of the 5,000 stations has some identifying information about the station and then a list of 31 weather observations. I=D indicates that the inversion option is being chosen for BSN, CITY, and REGION. This will cause the Datacomputer automatically to build inversion tables, which allow for content-based retrieval without sequential search of the data base.

Figure 6 shows a retrieval request that selects and outputs data based on the value of REGION and the

```
CREATE RESULTLIST PORT LIST
        RESULT STRUCT, P=EOR
                CITY STR (22)
                DATE STR (3)
                TEMPERATURE STRUCT
                        MIN STR (4)
                        MAX STR (4)
                        END
                END
;
```

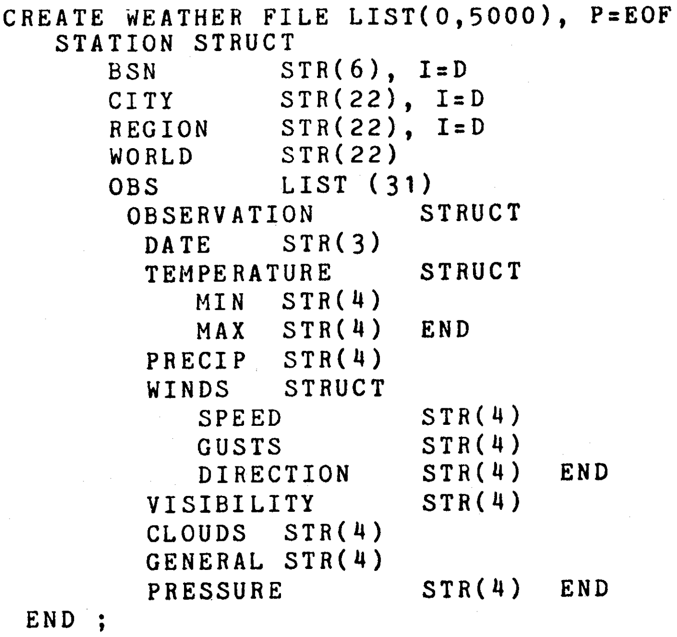
Figure 4—Sample datalanguage port descriptioı

maximum temperature.* The for-loop selects those stations with REGION equal to Massachusetts. Since the inversion option was chosen for REGION in the file description, the Datacomputer does not actually look at each station, but uses the inversion to find the selected stations. However, the user program submitting the retrieval request need not know that REGION is inverted; the request could be executed in any case.

For each of the selected stations, the second for-loop retrieves observations with TEMPERATURE.MAX greater than 300 (degrees Kelvin). Transmittal of data is indicated by assignment. Each RESULT record has four values: CITY, DATE, TEMPERATURE.MAX, and TEMPERATURE.MIN.

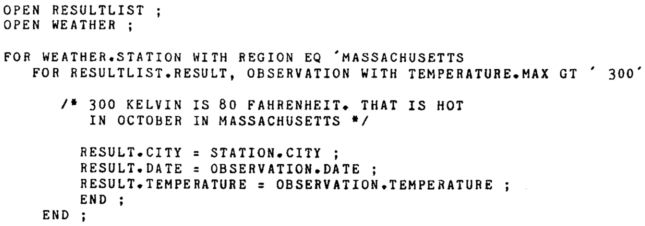
This example maps data from a 2-level tree-structured file to a 1-level tree-structured port. The observations in the port, unlike the ones in the file, are not organized by station; rather, the CITY is repeated for each output record.

In order for a request submitted by another machine to be executed, the Datacomputer must synchronize with external processes. Figure 7 shows the same request as above, along with the messages needed for synchronization of the Datacomputer and the other process. The first five characters are coded to be machine-processable. For example, the .I200 message indicates that the Datacomputer is ready for more datalanguage. Other messages direct the user program to send data, to send a new request, to close out the transactions, etc.



```
CREATE WEATHER FILE LIST(0,5000), P=EOF
   STATION STRUCT
      BSN         STR(6), I=D
      CITY        STR(22), I=D
      REGION      STR(22), I=D
      WORLD       STR(22)
      OBS         LIST (31)
       OBSERVATION        STRUCT
        DATE     STR(3)
        TEMPERATURE       STRUCT
           MIN   STR(4)
           MAX   STR(4)   END
        PRECIP   STR(4)
        WINDS    STRUCT
           SPEED          STR(4)
           GUSTS          STR(4)
           DIRECTION      STR(4)   END
        VISIBILITY        STR(4)
        CLOUDS   STR(4)
        GENERAL  STR(4)
        PRESSURE          STR(4)   END
 END ;
```

Figure 5—Sample datalanguage file description



```
OPEN RESULTLIST ;
OPEN WEATHER ;

FOR WEATHER.STATION WITH REGION EQ 'MASSACHUSETTS
   FOR RESULTLIST.RESULT, OBSERVATION WITH TEMPERATURE.MAX GT ' 300'

      /* 300 KELVIN IS 80 FAHRENHEIT. THAT IS HOT
         IN OCTOBER IN MASSACHUSETTS */

        RESULT.CITY = STATION.CITY ;
        RESULT.DATE = OBSERVATION.DATE ;
        RESULT.TEMPERATURE = OBSERVATION.TEMPERATURE ;
        END ;
    END ;
```

Figure 6—Sample datalanguage retrieval request

STATUS OF DEVELOPMENT

The Datacomputer has been offering service on the Arpanet since late 1973, using disk-storage only. Installation of the TBM tertiary store is scheduled for 1975.

The system is undergoing a phased development; successive versions offer increased capabilities to users by providing increasingly larger subsets of datalanguage. Thus, the development proceeds in an operational setting in which design errors and implementation bugs can be discovered early through feedback from actual users.

The version of the system currently offering service on the Arpanet (Version 0/11) is an intermediate version which provides adequate facilities for many applications, such as the ones described below, but by no means for all applications. An enhanced version is scheduled for mid-'75, with additional capabilities planned beyond that date.

As successive versions extend the range of datalanguage, previously written user programs incorporating datalanguage can either remain invariant or may require small modifications. Changes to Datacomputer hardware, such as the installation of TBM, are not reflected in datalanguage, and therefore require no change to user programs.

APPLICATIONS

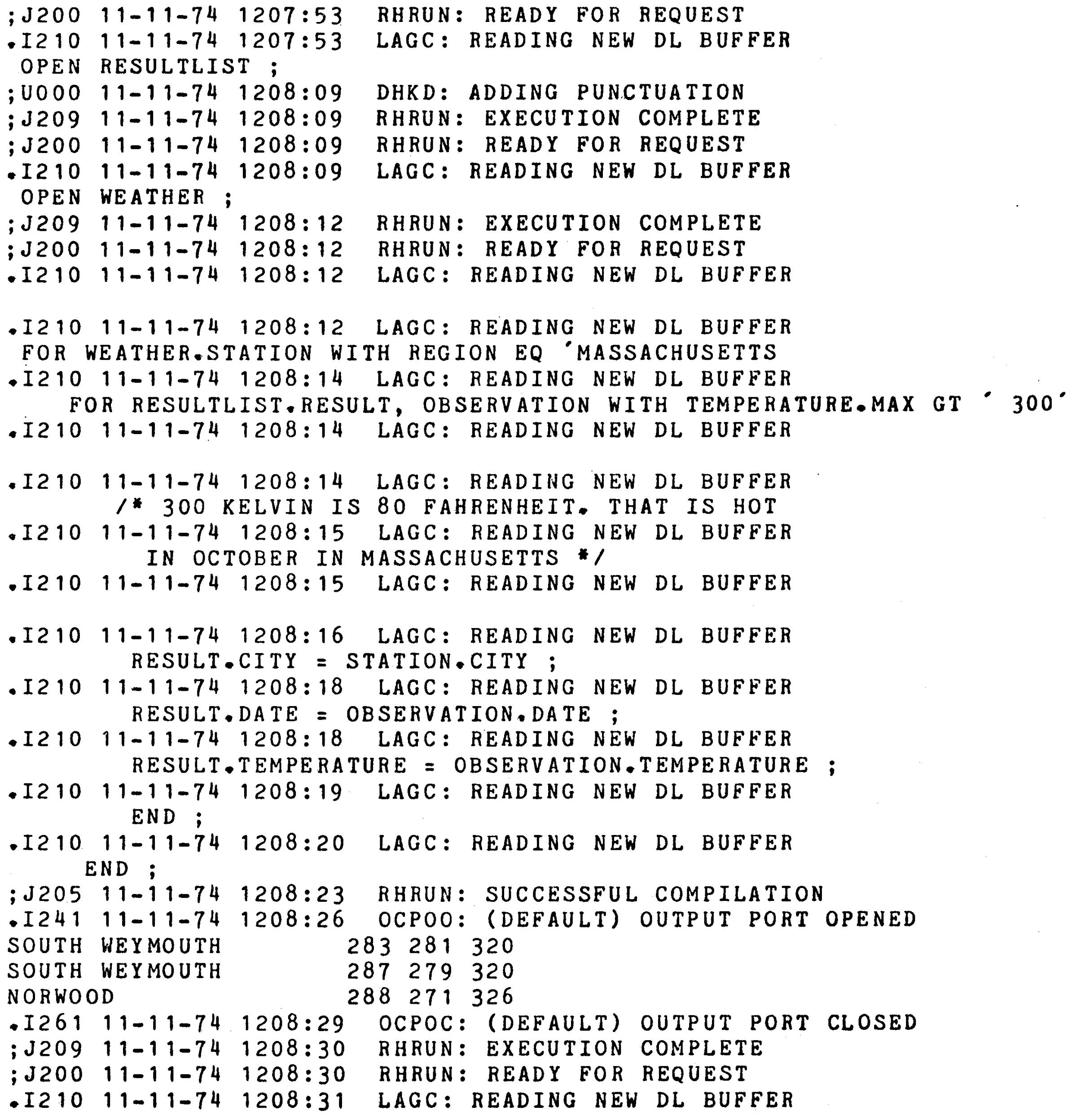
In this section three representative applications of the Datacomputer are discussed. The first two are in operation, and the third is currently being developed.

On-line information retrieval

As a service to the Arpanet community, a program at MIT Project MAC automatically surveys the status of all Arpanet hosts three times per hour around the clock. At each run, the SURVEY program attempts to connect up to each host, and stores the data, time, status, and response time. The data is automatically passed to the Datacomputer, where the historical SURVEY file is then updated by the current data.

As a companion to the data-collection facility, SURVEY provides on-line user functions that allow the database to be interrogated. A user on the network logs into MIT and composes his request for information in the on-line language supplied as part of the SURVEY application. The SURVEY program translates these requests into datalan-

* The symbols "/*" and "*/" are delimiters for comments.



```
;J200 11-11-74 1207:53  RHRUN: READY FOR REQUEST
.I210 11-11-74 1207:53  LAGC: READING NEW DL BUFFER
 OPEN RESULTLIST ;
;U000 11-11-74 1208:09  DHKD: ADDING PUNCTUATION
;J209 11-11-74 1208:09  RHRUN: EXECUTION COMPLETE
;J200 11-11-74 1208:09  RHRUN: READY FOR REQUEST
.I210 11-11-74 1208:09  LAGC: READING NEW DL BUFFER
 OPEN WEATHER ;
;J209 11-11-74 1208:12  RHRUN: EXECUTION COMPLETE
;J200 11-11-74 1208:12  RHRUN: READY FOR REQUEST
.I210 11-11-74 1208:12  LAGC: READING NEW DL BUFFER

.I210 11-11-74 1208:12  LAGC: READING NEW DL BUFFER
 FOR WEATHER.STATION WITH REGION EQ 'MASSACHUSETTS
.I210 11-11-74 1208:14  LAGC: READING NEW DL BUFFER
    FOR RESULTLIST.RESULT, OBSERVATION WITH TEMPERATURE.MAX GT ' 300'
.I210 11-11-74 1208:14  LAGC: READING NEW DL BUFFER

.I210 11-11-74 1208:14  LAGC: READING NEW DL BUFFER
       /* 300 KELVIN IS 80 FAHRENHEIT. THAT IS HOT
.I210 11-11-74 1208:15  LAGC: READING NEW DL BUFFER
         IN OCTOBER IN MASSACHUSETTS */
.I210 11-11-74 1208:15  LAGC: READING NEW DL BUFFER

.I210 11-11-74 1208:16  LAGC: READING NEW DL BUFFER
        RESULT.CITY = STATION.CITY ;
.I210 11-11-74 1208:18  LAGC: READING NEW DL BUFFER
        RESULT.DATE = OBSERVATION.DATE ;
.I210 11-11-74 1208:18  LAGC: READING NEW DL BUFFER
        RESULT.TEMPERATURE = OBSERVATION.TEMPERATURE ;
.I210 11-11-74 1208:19  LAGC: READING NEW DL BUFFER
        END ;
.I210 11-11-74 1208:20  LAGC: READING NEW DL BUFFER
     END ;
;J205 11-11-74 1208:23  RHRUN: SUCCESSFUL COMPILATION
.I241 11-11-74 1208:26  OCPOO: (DEFAULT) OUTPUT PORT OPENED
SOUTH WEYMOUTH        283 281 320
SOUTH WEYMOUTH        287 279 320
NORWOOD               288 271 326
.I261 11-11-74 1208:29  OCPOC: (DEFAULT) OUTPUT PORT CLOSED
;J209 11-11-74 1208:30  RHRUN: EXECUTION COMPLETE
;J200 11-11-74 1208:30  RHRUN: READY FOR REQUEST
.I210 11-11-74 1208:31  LAGC: READING NEW DL BUFFER
```

Figure 7—Sample datacomputer output and protocol messages

guage, sends the datalanguage to the Datacomputer, receives output from the Datacomputer, and presents the output to the user at his on-line terminal. The database management functions are all performed at the Datacomputer.

File management

A university computer center on the Arpanet routinely uses the Datacomputer system in a file management application, by means of a program called Datacomputer File Transfer Program (DFTP), which runs at the computer center. This program allows a local user program to store a file on the Datacomputer, retrieve a file, and add and delete a directory node. All DFTP-Datacomputer dialogue (datalanguage and protocol messages) is invisible to the user; the operation is automatic; access control mechanisms are provided. DFTP is particularly useful in this situation because the computer center is short of online storage for its users, and alternative solutions would involve magnetic tape and manual intervention.

Large shared file with multi-host access

In an application under development, the Datacomputer will be used as a central storage location and distribution point for a large database of seismic data collected from around the world in real time. Data will flow through the Arpanet to the Datacomputer, where it will be stored on-line. The data rate into the Datacomputer will grow over time, reaching a maximum of about 20 kilobits per second, 24 hours per day (6.3×10^{11} bits/year). Users of the data will be able to access the central database from any host machine in the Arpanet. By sending proper datalanguage requests to the Datacomputer, the host machine will be able to select arbitrary subsets of the large file and have these subsets shipped back in formats suitable for the particular host.

ACKNOWLEDGMENTS

Many people in the Datacomputer group at CCA have contributed to the development of the system, and their contribution is gratefully acknowledged. Special thanks are due to Richard A. Winter, Hallam G. Murray, David W. Shipman and Jeffrey M. Hill.

REFERENCES

1. Marill, T. and L. G. Roberts, "Toward a Cooperative Network of Time-Shared Computers," *Proceedings AFIPS Fall Joint Computer Conference,* 1966, pp. 425-431.
2. Roberts, L. G. and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *Proceedings AFIPS Spring Joint Computer Conference,* 1970, pp. 543-549.
3. Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," *Proceedings AFIPS Spring Joint Computer Conference,* 1970, pp. 551-567.
4. *CODASYL—Data Base Task Group Report,* ACM, New York, October 1969 and April 1971.
5. Gentile, R. B. and J. R. Lucas, "The TABLON Mass Storage Network," *Proceedings AFIPS Spring Joint Computer Conference,* 1971, pp. 345-356.
6. Damron, S., J. R. Lucas, J. Miller, E. Salbu, and M. Wildman, "A Random Access Terabit Magnetic Memory," *Proceedings AFIPS Fall Joint Computer Conference,* 1968, pp. 1381-1387.
7. Codd, E. F., "Recent Investigations in Relational Data Base Systems," *IBM RJ,* 1385, April 1974.
8. *Model 204 Database Management Software System-User Language Reference Manual,* Computer Corporation of America, September 1974.
9. Canaday, R. H., R. D. Harrison, L. L. Ivie, J. L. Ryder, L. A. Wehr, "A Back-End Computer for Data Base Management," *Communications ACM,* 1974, pp. 575-582.

RISS—A generalized minicomputer relational data base management system

by DENNIS McLEOD
Massachusetts Institute of Technology
Cambridge, Massachusetts

and

MONTE MELDMAN
Forest Hospital
Des Plaines, Illinois

INTRODUCTION

With the recent growth in popularity of low cost, relatively powerful minicomputers, it is clear that associated data base management systems are required. Nearly all of the minicomputer systems that are currently available commercially have at most rudimentary data base management capabilities. Consequently, in this paper we discuss the design and implementation of a minicomputer relational data base management system: the Relational Inquiry and Storage System (RISS). RISS provides a "naive" user interface, to allow nonprogrammers to deal routinely with a data base without the aid of a programmer, as well as an applications program interface. These interfaces facilitate data base access, modification, and restructure. RISS is a generalized and context-adaptable system, but it maintains a degree of efficiency guaranteeing cost-effective operation.

Minicomputer data base management

In this paper we are concerned with describing effective data base management facilities for computers used by organizations that have small scale needs and a small budget. There are many organizations which have application environments that require data base management capabilities, but which cannot justify a large expenditure of funds on a computer system. Minicomputers go a long way toward solving the hardware cost problem. But, more significantly, it is clear that such an organization cannot afford to develop the required computer system software *de novo*. Unlike the approach of Taylor and Lloyd[1] (for example), in which a complete minicomputer information system is developed, we will assume that a presupplied operating system and language translator (or translators) will be used. An example of such a system, which we will use in this paper, is the Resource Time-Sharing System (RSTS-11) for the PDP-11 computer supplied by the Digital Equipment Corporation.

In addition, it would be extremely desirable to permit an organization to obtain (e.g., purchase) a generalized data base management system which "fits neatly on top of" the operating system. Whitney[2] refers to this type of data base management facility as a fourth generation data management system. Some general considerations appropriate to the development of such a system have been reviewed by McLeod.[3] Since it is probably not possible to construct a generalized data base management system which can adapt to every aspect of a particular application environment, it is necessary to provide an effective interface between the data base management system and applications programs. But, whenever possible, the data base management system should allow nonprogrammers to deal directly with the data base. A large percentage of a user's needs should be satisified without the need to consult a programmer. Therefore, it is necessary to develop a very high level (nonprocedural) language to allow the "naive" user to deal effectively directly with the data base.

Approaches to data base management

As pointed out by Codd and Date,[4] there are two major approaches to data base management:

1. The network approach, as proposed by the CODASYL Data Base Task Group[5]
2. The relational approach, as proposed by Codd,[6] among others

A continuing controversy exists between the proponents of the network approach (e.g., Bachman[7]) and the proponents of the relational approach (e.g. Codd). A detailed analysis and comparison of these two approaches is presented by Codd and Date.[4,8] We believe that the relational approach is best suited to our goals and requirements. In support of the relational approach, Boyce, Chamberlin, King, and Hammer[9] state that "Traditionally, files are structured to optimize a particular application program. . . . A modern data base management system should be capable of responding to any new unanticipated query in uniform time without requiring a restructuring of the data and without impacting previously written queries." Specifically, for our purposes some of the most relevant advantages of the relational approach over the network approach are:

1. It provides a simpler and more unified user data model, resulting in systems that are easier to use and maintain.
2. It is much more data independent, and consequently results in systems that are more generalized. In addition, relational data bases are easier to alter, e.g., when new data relationships are discovered.
3. It is much easier to express data integrity constraints (limitations on the permissible data in a data base) in a generalized manner.
4. Data retrieval and modification requests are easier to express (in a generalized manner). These requests may be expressed in a way that is much less procedural.
5. The emphasis is on the use of sets (in the mathematical sense, not the CODASYL Data Base Task Group sense), rather than on handling one record at a time.
6. Sharing and protection requirements are more easily satisfied, due primarily to the simplicity of the underlying data base model and absence of highly distributed access paths.
7. Implementation issues are isolated from the logical data base model. This results in increased intersystem compatibility and, most significantly, allows a structured approach to implementation.

THE RELATIONAL APPROACH

Codd[6] introduced the relational model of data "which appears to be the simplest possible data structure consistent with the semantics of information and which provides a maximum degree of data independence."[9] As very concisely stated by Codd:[10] "In the relational approach there exists an interface at which the totality of formatted data in a data base can be viewed as a collection of nonhierarchic relations of assorted degrees defined on a given collection of simple domains (domains whose elements are not decomposable as far as the data base management system is concerned)."

In an attempt to apply some of the recent developments in the area of relational data bases to the minicomputer environment, it was decided to design a minicomputer relational data base management system. We have also implemented this system, calling it the Relational Inquiry and Storage System (RISS).

For any particular computer installation, RISS may be used to support one or more data bases. As an example, let us focus on one data base in the system, called "personnel information." A data base consists of relations and domains. One of the relations in this particular data base is called "employee," and is described below by a table representation:

Name	Number	Department	Sex
K. Smith	11135	Research	f
N. Greenberg	11136	Research	f
R. Jones	1125	Data Processing	m

The rows of the table correspond to tuples of the relation (records), and the columns correspond to instances of particular domains of the data base. A domain is a particular class of data values (objects). More than one column of a relation may have the same domain. A useful representation of a domain is a datatype. A datatype may be viewed as a specification of the class of objects a domain may contain. For example, the datatype "Sex" may be specified by stating that there are three possible values: "f" (or "female"), "m" (or "male"), and null (unknown). Of course, a datatype specification will often be more complicated than this.

The "employee" relation above has four columns. Each column is a distinct domain in this case. Each of the three tuples in this relation contains one data item (possibly null) for each column of the relation. Implicit here is the fact that tuples may be added to and deleted from the relation as "employees" are added or deleted. Also, the value of any column for a given tuple may be altered at any time. The structure of a relation may also be changed, e.g. by adding a new column; in some sense this results in a substantial change in the meaning of the relation.

RISS employs, without loss of generality, normalized relations,[11] so that they may be represented by a table. The table representation of a relation should not contain essential ordering information.[6] The table may be ordered in a useful manner, but the order should be reconstructable from data in the relation. Each tuple of the relation may be distinguished by its primary key, that is, by one or more columns which uniquely specify that tuple. In the "employee" relation, either "Name" or "Number" may be used as a primary key.

INTERFACE LEVELS

Liskov and Zilles[12] have discussed the use of abstract data types (levels of abstraction), and Aiello[13] has reviewed the support provided for this by existing programming languages. We believe that this approach to data management is one that has great potential value in conjunction with data base management systems. In addition, it seems advantageous to provide a structured (layered) approach to the data base management system interface. In this approach, each type of user is allowed to approach the system at a level consistent with his current needs.

In RISS, there are two interface levels. First, the "naive" user interface level is for routine (trained) users of the system, such as data clerks, as well as for managers and researchers who use the system occasionally. This interface level is based on a set of commands and command groups (cohesive sets of commands) which enable the "naive" user to access and modify the contents of a data base. Second, the (applications) program interface level allows programs to access and modify a data base by calling upon a set of primitive functions. Let us now examine the two interface levels in more detail. (Additional details may be found in the current RISS documentation.[14,15])

RISS functions

The RISS functions are a set of callable routines which may return information to the caller. These functions are callable from programs written in the language used to implement RISS, which is the same language that is used to write RISS applications programs. In effect, these functions define an extension of the (host) language. Functions are provided to facilitate the creation and deletion of relations to/from a data base, the addition and deletion of columns to/from a relation, the return of information about the columns (domains) of a relation, and the return or alteration of the value of a specified column for a particular tuple in a relation.

At this point, it is important to observe that in RISS we have not dealt with domains in the general sense that is most desirable. That is, we have not permitted the user to express a variety of types of integrity constraints. The ways in which the user may specify and/or restrict the permissible objects in a given domain are indeed very limited. The only type of constraint allowed is the specification of the storage representation, such as an integer or a varying length character string. We believe that it would be extremely advantageous to allow many other types of constraints such as "all the values of a column must be selected from a prespecified finite set of objects", or "all values in the 'name' column of relation 'employee' must exist in the 'person' column of relation 'master list.' " We are currently investigating the needs of the users of RISS in this area, as well as the applicability of abstract data types to the handling of domains.

Data base query languages

At this point, we should note that Codd[10] identifies five types of languages which have been developed for the query and modification of relational data bases:

1. element-by-element
   In this type of language only one tuple of a relation may be referenced at a time. The RISS (applications) program interface level is an example of this type of language.
2. algebraic
   These types of languages, such as MACaims[16] and the Multics RDMS[17] involve operations on entire relations.
3. relational calculus
   Codd's ALPHA,[18] the primary example here, is a language based on the calculus of relations.
4. mapping oriented
   SQUARE by Boyce, Chamberlin, King, and Hammer[9] provides an example of this type of language. In SQUARE the user learns a simple query specification format which is powerful and concise. An English language oriented version of SQUARE has been implemented by Boyce and Chamberlin.[19] Query by Example by Zloof[20] is another rather ingenious example of this type of language. The RISS "naive" user interface level is related to these languages, although it is certainly less general.
5. natural language
   Codd's RENDEZVOUS[21] can truly deal with "casual" users (users who have little knowledge of the data base management system), by allowing them to express queries in natural language. The system will ask the user for help if it cannot completely understand the query. Clearly this type of language is extremely desirable, but nontrivial to implement.

As mentioned previously, we believe that it is advantageous to utilize a structured approach to the relational data base management system interface. A good approach might be a four level system, such as:

1. a set of element-by-element primitives (lowest level), such as the RISS primitive functions
2. a set of operations based on the relational calculus, such as ALPHA
3. a language, such as SQUARE, for "naive" users (or the RISS "naive" user interface level)
4. a natural language based level for "casual" users, such as RENDEZVOUS

As a compromise, RISS has levels of the first and third types only.

RISS "naive" user interface level

The RISS "naive" user interface level has three basic components, which we will discuss in some detail: a relation editor, a retrieval package, and a data base manipulation and maintenance package.

The editor

It seems very important that users who perform editing operations on relations (such as creating and updating tuples) be provided with a cohesive package which supports their needs but isolates them from unnecessary details. RISS provides such a package: the RISS editor. The user enters the editor, specifying the relation to be edited, and may then use the editor commands to examine and modify the relation.

The editor design is based on that of the Multics editor "edm."[22] A relation is viewed as an ordered list of tuples. There is a "current tuple pointer", which points to the first tuple in the relation when the editor is entered, and may be moved by editor commands so that it points to any tuple in the relation. The editor contains commands that allow the user to:

1. move the pointer an integer number of tuples forward or backward
2. move the pointer by searching a column for a specified value (e.g. for character strings an "exact match" or "substring" search)
3. delete one or more tuples after (including) the current tuple
4. create a new tuple after the current tuple
5. display or change the value of a column of the current tuple (several ways are provided), or of all tuples in the relation
6. provide descriptive information about the relation (e.g. the (domain) name of a column)

The retrieval package

The RISS retrieval package is designed to allow the user to retrieve and analyze the data in RISS relations. It provides commands which facilitate:

1. selection of a set of retrieved tuples (retrieved set) on the basis of a column value comparator (e.g. sex="male", age>18)
2. modification of the retrieved set by forming the union or intersection of the retrieved set with a new set of tuples selected by a column value comparator
3. extraction of a subset of the columns of the tuples of the retrieved set
4. printing a tabular or other type of report based on the retrieved set
5. printing simple statistical information (e.g. mean, median, for numerical data) for the retrieved set
6. forming several groups of the data in a particular column by specifying a range of values for each group, and obtaining a list of how many tuples of the retrieved set fall into each group
7. producing a list of all the distinct values of a particular column for the retrieved set and the frequency of occurrence of each distinct value.

Of course, complex combinations of the above operations are also possible.

The maintenance and manipulation package

The RISS data base maintenance and manipulation package allows the user to perform various operations necessary for the maintenance and routine use of a data base. Not all of these operations need be accessible to all users. They include facilities for relation creation and deletion, relation copying, sorting, and merging ("joining"), adding and deleting columns to/from a relation, etc.

IMPLEMENTATION

An implementation of RISS has been completed using the RSTS-11 computer system at Forest Hospital in Des Plaines, Illinois. The language used, both to implement RISS and to allow applications programs to access RISS (the host language), is called Basic-Plus. A relatively powerful translator for this high level language is supplied with the RSTS-11 system.

Although implemented in a high level language via a translator which is not a strict compiler but is partially interpretive, the RSTS-11 implementation of RISS is sufficiently efficient. The basis for this statement is the general opinion of the RISS users that the system is very responsive. This is not to say that there are not some "bottlenecks" in the initial version of RISS, but that in general these appear not to be serious with respect to the context of the minicomputer environment.

It may be interesting to mention that the RSTS-11 RISS implementation was accomplished in less than .4 man-years. Work began in February, 1974 and the initial version was completed and released for general use in August, 1974. Since its release, RISS has run with no modifications to the software. We believe that this satisfying result is due to the structured approach taken in the design and implementation (such as the structured interface). In addition, a surprisingly small amount of data base maintenance (by programmers or experts) has been required. That is, "naive" users have been able to maintain their own data bases for the most part.

Storage of information

All information necessary to support RISS data bases is stored in files provided by the RSTS-11 file system. The scheme adopted for RISS is to use three files for each relation. First, the relation descriptor file contains a description of the relation, including the number of tuples currently in the relation, along with a description of each column of the relation and the corresponding domain. Second, the tuple file contains an entry for each tuple in the relation. Each entry is a list of pointers (possibly null), one for each column of the relation. These entries are stored sequentially as fixed-length blocks in the current implementation. The pointers in the tuple file point to the actual data items, which are stored in the third file, the column value file. Actually, if the data item is capable of being stored in a space smaller than that occupied by a pointer (three words), then it is stored in place of the pointer. Additionally, there is a file that contains the names of all relations and domains in a data base (a data base "directory").

All files are accessed via the RSTS-11 mechanism which operates on a standard buffered file mechanism, with blocks of 512 bytes (ASCII characters). However, a method has been developed to allow the RISS functions (applications programs interface level) to deal with four datatypes: pointers, integers, floating point numbers, and variable length character strings. This was accomplished by constructing a set of file system interface functions which convert these four datatypes into a form compatible with the basic RSTS-11 file mechanism.

RISS functions

As discussed previously, the RISS functions operate on a low level, and unlike the "naive" user interface level, their basic structure is implementation dependent. Specifically, this is because they actually manipulate the relation representation. The standard set of RISS functions present in the RSTS-11 implementation deals with the relation file representation described in the previous section (above). In this scheme, tuples are referenced by their index (row number in the table representation), and columns in a similar manner (column number). If another relation representation is desired, or a modification of the simple table representation is desired, such as inverting one or more columns (e.g., the primary key), then a modified set of functions can be constructed. Such a method for inverting columns in RISS relations is currently under development.

It is important to note that, due to the limitations in the facilities for modular programming in RSTS-11, it was necessary to design a "preprocessor" for all RISS programs. This preprocessor scans a program for references to RISS functions, and inserts the code for the functions used to a temporary module, which is then compiled. Thus, a compiled module that performs no external calls (to access RISS functions) is produced.

Limitations

At this point, it seems appropriate to mention what appear to be some of the most significant limitations of the RISS system (Forest Hospital implementation):

1. The handling of domains is insufficient. Specifically, no general method has been provided for expressing domain integrity constraints, i.e., limitations on the types of admissible values (objects) in a domain.
2. Concurrent access to relations in a data base is limited, since the implementation thereof is based on the scheme provided by the host operating system. Many users are allowed to examine a relation simultaneously, but only one user may update that relation (a standard locking mechanism is employed).
3. Column naming may be ambiguous. Only one name is assigned to each column of a relation: the name of the underlying domain. Thus if two columns in a relation have the same name, they must be referenced by their column number. A unique name for each column in a relation should be added, and users should not have to remember column numbers.
4. Retrieval involving more than one relation is not simple. In order to formulate a query that involves more than one relation, it is necessary to explicitly create a temporary relation (via one or more joins) using all the relations involved in the query.
5. Protection is based on the limited scheme provided by the host operating system. Read and/or write permission may be given to groups of users, but further refinement of protection constraints (e.g., to columns of a relation) is not possible.
6. Since joins are represented as "snapshots" of a data base, it is possible for inconsistencies to arise between the relations and the (implicit) joins when updates occur. Consequently, a type of data base integrity is not strictly enforced.

Other limitations exist, but in the interest of brevity they will not be discussed here. In addition, a survey of the users of RISS is being made to determine what modifications to RISS are necessary and/or desirable.

CONCLUSIONS

It has been shown that a relational data base management system is practical in the minicomputer environment. Although some compromises in power and generality were necessary, a useful and cost-effective implementation has been produced. The implementation has been produced in a rather short amount of time and is surprisingly "self-maintaining". In addition, the implementation was accomplished using an existing minicomputer operating system and host language, and at a low development cost.

ACKNOWLEDGMENTS

The authors would like to thank Robert Pellicore of Forest Hospital for his helpful suggestions concerning the RISS applications program interface (level). Professors Michael Hammer and Barbara Liskov of Project MAC at the Massachusetts Institute of Technology have been helpful in providing useful comments on the subject matter of this paper. The referees have also made many useful comments on the content of the paper. Kathy Vick, Nancy Wingren, and Edie Johnson of Forest Hospital have provided many helpful comments on the RISS "naive" user interface level. Marsha Baker of the Department of Electrical Engineering at M.I.T. assisted in the preparation of the manuscript.

REFERENCES

1. Taylor, B. J. and S. C. Lloyd, "DUCHESS—A High Level Information System," *1974 National Computer Conference Proceedings,* Volume 43, AFIPS Press, Montvale, NJ, May, 1974, pages 35-40.
2. Whitney, K. M., "Fourth Generation Data Management Systems," *1973 National Computer Conference Proceedings,* Volume 42, AFIPS Press, Montvale, NJ, June 1973, pages 239-244.
3. McLeod, D. J., *Relational Data Management in Minicomputers,* M.I.T. Department of Electrical Engineering, S.B. Thesis, Cambridge, MA, February, 1974.
4. Codd, E. F. and C. J. Date, "The Relational and Network Approaches: Comparison of the Application Programming Interfaces," *Proceedings of 1974 ACM-SIGFIDET Workshop on Data Description, Access, and Control,* Ann Arbor, MI, May, 1974.
5. *CODASYL Data Base Task Group Report,* ACM, New York, NY, April, 1971.
6. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM,* Volume 13, Number 6, June, 1970, pages 377-387.
7. Bachman, C. W., "The Programmer as Navigator," *Communications of the ACM,* Volume 16, Number 11, November 1973, pages 653-658.
8. Codd, E. F. and C. J. Date, "Interactive Support for Non-programmers: The Relational and Network Approaches," *Proceedings of 1974 ACM-SIGFIDET Workshop on Data Description, Access, and Control,* Ann Arbor, MI, May, 1974.
9. Boyce, R. F., D. D. Chamberlin, W. F. King, III and M. M. Hammer, "Specifying Queries as Relational Expressions: SQUARE," *Proceedings of ACM SIGPLAN-SIGIR Interface Meeting,* Gaithersburg, MD, November 1973.
10. Codd, E. F., "Recent Investigations in Data Base Systems," *Information Processing '74,* North-Holland, Amsterdam, 1974.
11. Codd, E. F., "Normalized Data Base Structure: A Brief Tutorial," *Proceedings of 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control,* San Diego, CA, 1971.
12. Liskov, B. and S. Zilles, "Programming with Abstract Data Types," *Proceedings of a Symposium on Very High Level Languages,* Santa Monica, CA, March, 1974.
13. Aiello, J. M., *An Investigation of Current Language Support for the Data Requirements of Structured Programming,* MAC TM-51, Project MAC, M.I.T., Cambridge, MA, September, 1974.
14. McLeod, D. J., *RISS Programmer's Guide,* Forest Hospital, Des Plaines, IL, August, 1974.
15. McLeod, D. J., *RISS User's Guide,* Forest Hospital, Des Plaines, IL, November, 1974.
16. Goldstein, R. and A. Strnad, "The MACaims Data Management System," *Proceedings of 1970 ACM-SIGFIDET Workshop on Data Description and Access,* Houston, TX, November, 1970.
17. Goldman, J., *The Use of Computed Relations in a Set Theoretic Data Base,* M.I.T. Department of Electrical Engineering, S. M. Thesis, Cambridge, MA, June, 1973.
18. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," *Proceedings of 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control,* San Diego, CA, 1971.
19. Boyce, R. F. and D. D. Chamberlin, *Using a Structured English Query Language as a Data Definition Facility,* IBM Research Laboratory, San Jose, CA, RJ1318-#20559, 10 December, 1973.
20. Zloof, M. M., *Query by Example,* IBM Thomas J. Watson Research Center, Yorktown Heights, NY, RC4917-#21862, 2 July, 1974.
21. Codd, E. F., "Seven Steps to Rendezvous with the Casual Users," *IFIP TC-2 Working Conference on Data Base Management Systems,* April, 1974 (published as *Data Base Management,* Amsterdam, 1974).
22. Project MAC, M.I.T., and Honeywell Information Systems, *The Multiplexed Information and Computing System: Programmer's Manual,* Cambridge, MA, 9 September, 1973.

A multi-level relational system

by J. MYLOPOULOS, S. SCHUSTER and D. TSICHRITZIS
University of Toronto
Toronto, Ontario

INTRODUCTION

A relation can be conceptually viewed as a table of data.[1] The table's heading defines the relation's name, the column headings are the attribute names, and each row corresponds to an n-tuple of data values describing a single entity. The set of values which can be used in a column is called a domain. A relational data base is composed of a set of time varying relations inter-related through common domains.

The relational model of data has been proposed as a flexible user interface to a data base management system (DBMS). Data languages based on relations are less procedural and more powerful than their existing counterparts in currently available systems. However, they leave optimization and access path decisions to the system. For example, the DBMS must decide how to best use many common mechanisms like pointer arrays, data pools, inverted files, and semantic networks in a manner transparent to the user.

We describe in this paper the mechanisms used in the development of a prototype system called ZETA/TORUS. We present in a hierarchical fashion a set of facilities which ultimately lead to the implementation of the complete system. Most of the techniques used are present in other systems. The important consideration is to isolate and outline their role in implementing relations.

This system is composed of three principal levels. The low level system implements elementary relational operations, inverted files, and the ability to "mark" subsets of relations. The intermediate level implements derived relations. It also has the ability to combine elementary queries of the lower system into high level relation operations. Finally the high level uses a compiler-compiler for query language generation, a preprocessor/compiler for a host language system, and a semantic network for a natural language understanding system.

THE PRIMITIVE RELATIONAL LEVEL

At this level the system provides a basic facility to manipulate relations.[2] An "actual" relation can be created and is stored in a basic direct access file. Each tuple of the relation corresponds to a record of the file. Names of the relations and their domains and other characteristics are stored in system tables. Each one of the created relations has independent existence and it can be queried or updated through commands. All commands are procedure calls to this level of system. The format of the commands is very strict and users must abide by many conventions. The main purpose of this level is to implement some important mechanisms and not to provide a user interface.

The system provides a complete set of commands to deal with tuples within named relations. Namely the user has commands to:

(a) Create and destroy relations,
(b) Lock and unlock relations,
(c) Insert and delete tuples,
(d) Get and update data elements within a single tuple,
(e) Mark relations.

There are no commands applying to more than one relation at this level.

The system also provides a primitive data definition language. In addition, users are able to query, but not update, the schema tables and get the proper names for domains, relations, etc. This facility provides an elementary form of data dictionary.

The most interesting command of this level is the marking operation. A "mark" corresponds to a unary relation which stores indices of tuples of an actual relation which satisfy a Boolean qualification. We use this facility as a tool for the construction of more complex operations.[3]

The unary relation is simply an array of indices related in two ways:

(a) The indices point to tuples of the same "actual" relation,
(b) The tuples referenced satisfy the same qualification.

These unary relations are implemented using a header and a body. The header contains control information while the body contains the indices.

Several basic operations are implemented to manipulate these structures. The user may create or destroy marks. He can retrieve tuples of the marked relation. At this point, modification of marks by the user is not allowed. Future extensions will incorporate such facilities in order to provide some of the proposed functions of higher levels.

The mark operation provides a tool which can be used by the higher levels of the system to implement complex relational operations (e.g., joins). A mark operation on a mark is also allowed. This results in the creation of a subset of indices of a first mark according to a second qualification. A mark on the schema is also allowed where the schema is viewed as a relation. This way the users can isolate the parts of the schema for high level security or for data dictionary queries.

The qualification for a mark is passed as a binary tree of a Boolean expression of basic domain-comparitor-value triplets. A method for efficiently executing Boolean qualifications on partially inverted relations has been studied and is being incorporated into the system.[4]

THE INTERMEDIATE RELATIONAL LEVEL

This level is concerned with supporting high level relational data base structures. Two important functions are provided at this level: derived relations and multi-relation queries.[5] It provides the higher language level an interface which is suited to its user's goals and interfaces with the basic level in order to accomplish its tasks. The interface to this level is through a rigid data structure so that interpretation is straightforward.

The execution portion of this level must concern itself with the semantics of user requests with respect to the method used to implement relations. Relations that have been implemented as virtual constructs require different algorithms for executing the various types of operations that can be performed on them. Also, high-level optimization can occur whenever an operation or relation implementation can be accomplished by more than one method. The intermediate level is responsible for these decisions.

Derived relations

A primary relation is one that has been created by the data base administrator. A derived relation is a relation that is formed as the result of a join of two or more derived or primary relations or the result of a qualification giving a subset of a single relation's domains and tuples. Derived relations are an important component of a DBMS because users often need focus on a subset of the data base. More important, the re-execution of similar queries causing extensive retrievals are very expensive. ZETA allows a user to name and define a new relation with the same constructs used for qualifying and retrieving from an existing relation. Two types of derived relations play an important role in this system: snapshots and automatic derivations.

A snapshot is a time invariant "picture" of a portion of the data base at a particular instance. The resulting relation is no longer dependent on changes made to any of the relations involved in its creation. Such derivations are important as a journaling facility or because a stable environment is required. Such is required by programmable DBMS where operations tangential to retrievals are occurring to the data of a relation. If a snapshot were not created, then the original relation would have to be locked from updates perhaps for an uncomfortable amount of time.

Several implementation schemes have been considered for snapshots which define a subset of a single existing relation. It became clear that an optimal one depends upon the type of system usage that is encountered.

In a retrieval oriented environment (i.e., update is batched and performed when no retrieval is occurring), the marking scheme presented by the primitive level is perfectly adequate. An access to a derived relation in this configuration would require the indirect access through a pointer to a tuple. Since the relation is not stored as an "actual" relation it is called "virtual".

In a very active update environment, the great amount of overhead incurred by updating a pure marking scheme would make it prohibitive. Instead a snapshot could be formed by creating a separate actual relation—one whose tuples are the records of a separate file. This is accomplished by first locking update access to the relation, then by forming a marking on the qualified tuples of that relation, and finally by retrieving and copying the required domain values to the snapshot.

Both of the above schemes are being implemented. However, since a straight marking is not adequate in an update environment and because complete actual implementation is costly in terms of storage, a partially virtual compromise is proposed. A snapshot is originally implemented as a marking. Each time a change occurs to a tuple of the original relation, the old tuple is copied and logged into a separate section of the primary relations file. The mark that originally pointed to the actual tuple now points to its new location and the original tuple is updated. When the number of marks which points to the augmented primary relation reaches a threshold, then the snapshot is converted to an actual representation and the marks are destroyed.

An automatic derivation is a time-dependent relation which reflects the changes that have occurred to each of its ancestors. A snapshot would have to be re-created each time a user wants an updated version while an automatic relation remains "up-to-date". Several schemes have been proposed.[2]

An automatic relation is equivalent to a snapshot in a retrieval-oriented environment. Because of this, a marking is completely adequate. However, in this case the problem is of no interest because, by definition, updates are why we consider automatic derivations.

One simple scheme would be to save only the definition of the relation. When a query involving the derived relation is initiated, the relation is locked against other disturbances, the derived relation is formed from the stored definition, and then the relation is destroyed. This is, of course, an expensive procedure but may prove adequate in an environment where automatic relations are required only once in a while.

A third, intermediate scheme, conforms to the same philosophy as the partially virtual scheme presented for snapshots.[2] At first, the derived relation is formed as a marking and its definition is saved. This time, changes are logged by storing the new tuples as part of an augmented primary relation and marks pointing to the old tuples are made to point to the new tuples. These changes must be time stamped. The next time the derived relation is accessed, the definition is re-executed only on the tuples not checked since the last access.

It is clear from this discussion that derived relations must be used carefully. Snapshots must not be kept too long and automatic relations must not be used too often. Also, the incorporation of derivation schemes requires the formation of an intermediate level schema for the translation between logical operations and virtual implementations.

Multi-relation queries

A multi-relation operation or query requires the manipulation of data from two or more relations which can be implemented either as primary or derived relations. Two types of multi-relation queries arise in this system: join and composite.

The join operation creates a new relation by combining two existing relations. This is accomplished by comparing the relations on one or more domains that are common to each relation and by forming the new tuples by associating a tuple from one relation with one or more tuples of the other.

The composite operation (also called an implicit join or restriction) extends the qualification capability for selecting subsets of tuples of a relation. It allows the results or values retrieved from one relation to be used as qualification for the retrieval over a second relation.

System structure

The intermediate level system is composed of three principal components: the interpreter, a set of intermediate level schema procedures, and a set of utilities. All three components are embedded within a single external procedure in which the intermediate level schema tables and interface command data structures are global. This system is invoked by a call from the higher level system passing the command data structure as a parameter. The interpreter breaks the command down into a sequence of utility operations. The utility procedures interface with the primitive level system. Schema procedures are invoked by the interpreter whenever data pertaining to derived or primary relations are to be stored or retrieved.

THE USER INTERFACE LEVEL

Three major classes of users requiring access to DBMS can be described: application programmers, technical personnel, and casual users.

An application programmer requires a programmable DBMS language in order to access data to formulate complex queries, special reports, or perform atypical computation not provided by other interfaces. He must understand the organization of data but can often ignore the meaning of the data being processed.

A technical user is a person who specializes in the semantics of the data and does not want to concern himself with computer system details. Examples of such users are urban planners, doctors, administrators, management scientists, engineers, etc. Query languages have evolved to provide these technical users with an English-like language for interactive communication with a DBMS. The power of this type of interface is that the user does not have to seek an application programmer for most of his data processing and that he can "browse" through his data base by reformulating queries based on data just received.

The casual user is a person who either requires access to a DBMS so infrequently as not to warrant learning a high or low level programming language or who refuses to learn the discipline of a query language. The only dialogue acceptable is some form of natural language.

All three types of interfaces have been investigated for interacting with the lower levels of the relational DBMS.

A programming language interface

A programmable interface to this relational DBMS is being implemented by embedding a set of data manipulation language (DML) subroutines within a PL/1 host language environment.[5] The syntax for a qualification is a modified subset of SEQUEL.[6] Commands to the DBMS are passed as structured English-like character strings within a procedure call. A preprocessor has been designed so that a "cleaner" interface can be effected and so that syntactic errors can be flagged before execution of the PL/1 program.

The DML embodies the capabilities of the intermediate level system, thus making the translation process straightforward. Its capabilities provide:

(a) Multi-relation qualifications,
(b) Derived relations based on qualifications,
(c) Updates based on qualifications,
(d) Interaction with host program variables in order to express complex queries by:

(i) passing "generated" values for dynamic qualifications,
(ii) passing "generated" requests.

Since this language has all the capabilities of PL/1, it could potentially serve as a system implementation language for the development of language processors of the following two types.

A query language facility

The most important characteristic of a query language is how well it is tailored to its problem area. General purpose languages often force users to think in terms of unnatural logical data structures and syntax. The goal of this work is to construct an environment in which problem-oriented query language generation could be readily accomplished.

An existing syntax-table-driven compiler-compiler was modified to support a query language development facility.[7,8] This system allows users to easily specify the syntax of their language and to connect it to a set of modular semantic routines. The semantic routines utilize the intermediate level (which is accessible through PL/1) as a semantic language. This allows the symbol table and other system tables (which could include the tables of the syntactic network used by the fixed parser) to be stored, retrieved, and maintained by the DBMS.

Because the syntax of a users source language is specified in a table, subsets of a language can easily be generated by eliminating parts of the table. This provides several benefits:

(a) A customized user view; unused capabilities may be omitted,
(b) Security; e.g., by eliminating the UPDATE keyword for a specific user, the associated capability is unaccessible,
(c) Reduced compilation time; the number of optional syntactic constructs are reduced so that parsing time is shortened.

A special facility for macro definition and expansion was also added to the generator. The use of macros in a query language can provide the powerful capability of allowing one user to specify a complex query and have another utilize it easily. It also provides a much needed abbreviation facility for teletype terminal environments.

Because a specific application data base has not yet been explored, a general purpose relational query language was built using the facility. It embodied all the features of the intermediate level system and added:

(a) A sequence of language subsets,

(b) User macro definition,

(c) Report generation:

  (i) sorting,
  (ii) output device choice,
  (iii) titles and headings,
  (iv) position and type formatting,

(d) Data dictionary.

An intelligent natural language interface

The expected increase in the use of DBMSs will cause an ever increasing need for making DBMSs more readily available to the casual user. The aim of the TORUS (Toronto Understanding System) project is to achieve this goal by:

(a) Eliminating the need for the user to know *how* information is stored in the DBMS and only expecting him to know *what* is stored,
(b) Enabling him to communicate with the DBMS in simple English and without the arbitrary restrictions of programming languages.

The methodology we have chosen in order to achieve the TORUS aim is different from that of earlier attempts to provide a natural language interface with the user.[9,10,11] TORUS extensively uses semantic networks as a basis for "understanding" the ongoing dialog with a user as well as the information stored in the DBMS. This methodology was chosen because it is compatible with the philosophy that there are no shortcuts to the problem of making a system appear intelligent to a user who lacks the training and/or the time to program it. This intelligence can only be realized by a system which has and can use the knowledge relevant to the universe of discourse—in this case a data base—in ways which, at least abstractly, are similar to those used by humans. It follows from these premises that the representation and use of knowledge about a particular data base stored in a relational manner is the most important problem to be tackled in designing a natural language front end to a DBMS.

TORUS consists of four basic modules:

(a) INPUT: The input module accepts a sentence in English, performs a syntactic analysis, and outputs a tree structure which describes the sentence's underlying syntax.[12] This structure is then passed to the SEMANTICS modules.
(b) SEMANTICS: This module is responsible for "making sense" out of the structure passed by INPUT. This is achieved by using a semantic network and associated procedures, and results in a complete integration of the input sentence with the semantic network. If the input structure does not make sense, another parse is requested from INPUT. If there is a need for use of the DBMS, appropriate commands are passed to the INTERFACE module and execution of SEMANTICS is suspended until a message is returned by the interface. It is then decided what should be output and a structure similar to that constructed by INPUT is sent to the OUTPUT module.
(c) OUTPUT: The structure received from SEMANTICS is mapped on to a sentence,[13] using some previously explored ideas.[14,15]
(d) INTERFACE: This module receives lists of commands to update, retrieve, or test the validity of information in the data base. These lists are translated into sequences of commands for the DBMS and its responses are passed back to SEMANTICS.

The system is currently being tested with a data base involving student records. Each record stores general information about a student (place and date of birth, student number, marital status, etc.), his/her academic background (universities attended, courses completed, degrees obtained) and his/her record in the Department of Computer Science at the University of Toronto.

The functions of the system's modules will be illustrated by considering a simple sentence like "What is John Smith's address?" and describing how it will be handled by the TORUS system.

When INPUT is presented with the sentence, it first performs a lexical analysis of the words present. 'John' and 'Smith' have to be marked as proper nouns by the user who types in the question. The sentence is then passed to an augmented transition network parser which attempts to find a parse. It must be noted that the parser will not attempt to resolve semantic ambiguity problems (e.g., reference of prepositional phrases, selection of a word sense when there are several candidates in the same syntactic class as in "I took a course" and "I took a book", etc.).

The structure constructed by INPUT is then passed to the semantic module, which first attempts to construct a graph representing the meaning of the input sentence. This is done by first using "case frames"[17] which check whether the semantic relationships implied by the parsed structure are meaningful. Several "mapping functions" are then used to obtain a deeper representation of the meaning of the input sentence, e.g., 'take' may have a mapping function which will map the structure described by:

{take: agent=student, object=course,
source=professor}

which may have been obtained from

"The student took a course from the professor"

into the semantically deeper structure

{teach: agent=professor, object=course|
enrol: agent=student, destination=course}

There is no need for mapping functions for the sample sentence.

Anaphoric reference problems are also treated at this point. The function which attempt to solve such problems will take into account semantic properties of the object referred to and will attempt to identify it on that basis.

Another job that has to be done at this time is to determine the type of action (i.e., retrieval, update, count, test-for-validity) that will have to be performed by the system. In this case the action is 'retrieve'. Note that the sentence "Give me John Smith's address" will be mapped into the same graph as that of our current example, 'give' serving only as an indicator that it is the 'retrieval' command that is applicable here.

The next step involves "fitting" the constructed graph to the semantic network which stores general knowledge the system has about student records as well as specific information it has obtained during the current conversation either from the data base or the user. In the network there will be nodes (concepts) which correspond to our general notion of 'address' (it is something that can characterize a person or an organization) as well as that of 'current-address'. Moreover, 'current-address' will be represented as a sub-concept of 'address' and it will be associated with an attribute of the relational data base named CURRENT-ADDRESS where a student's current address is stored. Thus the semantics of the attribute CURRENT-ADDRESS of the relational data base is defined by the properties of the concept 'current-address' of the semantic network.

During the fitting of the graph, "recognition functions" are often used to determine whether a node of that graph represents an instantiation of a generic concept on the semantic network. A recognition function can use common-sense knowledge about a generic concept which might help it decide that 'abc def' is not an address, but it may also check the input sentence and the semantic network to make inferences about which nodes of the input graph are instantiations of a particular generic concept.

Once the graph has been fitted on the network, the system knows that something must be retrieved from the data base (since a 'retrieve' command was generated) and where to find it (since the 'address' node of the input graph was found to be an instantiation of the 'current-address' concept to which the attribute CURRENT-ADDRESS is associated). A message is therefore passed to the INTERFACE.

(GET 1; ADDRESS=?; NAME=John Smith)

which asks for the retrieval of a single item from the data base that is consistent with the information given (the name) and includes the missing information (the address). The interface will return

(ADDRESS=65 Charles St., Toronto;
NAME=John Smith)

and SEMANTICS will then place this information on the semantic network.

Finally, SEMANTICS decides what portion of the semantic network should be output by taking into account the input sentence and the information that was retrieved from the data base. The structure constructed is quite similar to that created by INPUT, except that the question mark has been replaced by '65 Charles St., Toronto'. Several "inverse-mapping functions" may be used during the construction of this structure, mapping the deeper representation of the sentence to be output into a more surface form. The OUTPUT module now constructs a string by using an augmented transition network.

Several features we consider important have been implemented superficially in the current version, and others are missing completely. Thus the system only handles simple cases of conjunction, disjunction and quantification. Moreover, it has limited inference capabilities and its "understanding" of the ongoing dialog is accordingly restricted.

CONCLUSION

The ZETA data base management system is implemented in PL/1. The primitive level is completely designed, implemented and tested. The intermediate level and query system level has been designed and partially implemented. We are finishing implementation and integration at this point. A report with the status of the project is forthcoming.

A first version of TORUS, which handles simple sentences and contains only a few mapping, inverse mapping, and recognition functions has been implemented and is currently being tested. The languages used for the implementation are SPITBOL[20] and 1.PAK an AI language offering directed labelled graphs and graph pattern matching.[21] The details of the current version can be found in a technical report.[22]

ACKNOWLEDGMENT

We would like to acknowledge the contributions of the following students and colleagues who made the ZETA/TORUS project a reality: Michael Brodie, Simone Chan, Bryon Czarnik, Gilles Farley, Jack Klebanoff, Edwin Leong, Ellen Poon, Olga Puchmajerova, Dick Swenson, Alda Usas for ZETA; Alex Borgida, Phil Cohen, Nick Roussopoulos, John Tsotsos, Lindsay Watt and Harry Wong for TORUS.

This research was supported in part by the National Research Council of Canada, the Department of Communications of Canada, and by IBM Canada Ltd.

REFERENCES

1. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *CACM,* June 1970.
2. Czarnik, B., *A Primitive Relational Data Base Management System,* M.Sc. Thesis, Department of Computer Science, University of Toronto, October 1974.
3. Tsichritzis, D. C., "On Implementation of Relations," *Proceedings of the ACM SIGFIDET Workshop,* Ann Arbor, Michigan, May 1974.
4. Farley, J. H. G., *Query Evaluation on Partially Inverted Relations,* M.Sc. Thesis, Department of Computer Science, University of Toronto, September 1974.
5. Leong, E., *A Host Language Relational Data Base Management System,* M.Sc. Thesis, Department of Computer Science, University of Toronto, September 1974.
6. Chamberlin, D. and R. Boyce, "SEQUEL: A Structured English Query Language," *Proceedings of the ACM SIGFIDET Workshop,* Ann Arbor, Michigan, May 1974.
7. Gaffney, J., *TACOS: A Table Driven Compiler-Compiler System,* Department of Computer Science Report 325, University of Illinois, June 1969.
8. Chan, S., *QLS: A Query Language Generator System,* M.Sc. Thesis, Department of Computer Science, University of Toronto, October 1974.
9. Woods, W. A., "Progress in Natural Language Understanding—An Application to Lunar Geology," *Proceedings NCC,* November 1973, pp. 441-449.
10. Kellogg, C., J. Burger, T. Diller, K. Fogt, "The CONVERSE Natural Language Data Management System: Current Status and Plans," *Proceedings of the Symposium on Infor. Storage and Retrieval,* 1971, pp. 33-46.
11. Codd, E. F., "Seven Steps to Rendezvous with the Casual User," *Proceedings IFIP TC-2 Working Conference on Data Base Management Systems,* Cargese, Corsica, April 1974.
12. Borgida, A., *Topics in the Understanding of English Sentences by Computer,* M.Sc. Thesis, Department of Computer Science University of Toronto, November 1974.
13. Wong, H., *Generating English Sentences from Semantic Structures,* M.Sc. Thesis, Department of Computer Science, University of Toronto, January 1975.
14. Simmons, R. F., "Semantic Networks: Their Computation and Use in Understanding English Sentences," in Schank, R. C. and Colby, K. M. (Eds.) *Computer Models of Thought and Language,* Freeman and Co., 1973.
15. Simmons, R. F., J. Slocum, "Generating English Discourse from Semantic Networks," *Comm. ACM,* Vol. 15, 10.
16. Woods, W. A., "Transition Network Grammars for Natural Language Analysis," *Comm. ACM,* Vol. 13, 10, pp. 591-606.
17. Bruce, B., "Case Structure Systems," *Proceedings Third International Joint Conference on Artificial Intelligence,* August 1973, pp. 364-371.
18. Rumelhart, D., P. Lindsay, D. Norman, "A Process Model for Long Term Memory," in Tulving, E., Donalson, W. *Organization of Memory,* Academic Press, 1972.
19. Martin, W. A., Memos 6, 8, 11, 12, 13, Automatic Programming Group, MIT.
20. Dewar, R., *SPITBOL* Illinois Institute of Technology February 1971.
21. Mylopoulos, J., N. Badler, L. Melli, N. Roussopoulos, "1.PAK: A SNOBOL-based programming language for AI applications," *Proceedings Third International Joint Conference on AI,* August 1973, pp. 591-596.
22. Mylopoulos, J., A. Borgida, P. Cohen, N. Roussopoulos, J. Tsotsos, H. Wong, *The TORUS Project: A Progress Report,* Technical Report, Department of Computer Science, University of Toronto (To appear).

INGRES—A relational data base system*

by G. D. HELD, M. R. STONEBRAKER and E. WONG
University of California
Berkeley, California

INTRODUCTION

INGRES (Interactive Graphics and Retrieval System) is a relational data base and graphics system which is being implemented on a PDP-11/40 based hardware configuration at Berkeley. INGRES runs as a normal user job on top of the UNIX[1] operating system developed at the Bell Telephone Laboratories. The only significant modification to UNIX that INGRES requires is a substantial increase in the maximum file size allowed. This change was implemented by the UNIX designers. The implementation of INGRES is primarily programmed in "C", a high level language in which UNIX itself is written. Parsing is done with the assistance of YACC,[2] a compiler-compiler available on UNIX.

The advantages of a relational model for data base management systems have been eloquently detailed in the literature,[3,4,5] and hardly require further elaboration. In choosing the relational model, we were particularly motivated by (a) the high degree of data independence that such a model affords, and (b) the possibility of providing a high level and entirely procedure free facility for data definition, retrieval, update, access control, support of views, and integrity verification.

In this paper we shall describe most of the principal components of INGRES. These include: the query language QUEL, an algorithm for processing interactions based on the principle of "decomposition," the access methods supported, and an approach to access control, views, and integrity preservation via query modification. Not described in this paper are: control of concurrent updates, the graphics facilities, the system utility commands, and a more "friendly" graphics oriented user interface. These topics are presented respectively in References 6, 7, 8 and 9.

BASIC CONCEPTS AND DEFINITIONS

Let $D1$, $D2$, . . . , Dn be nonempty sets, not necessarily distinct. A subset R of the product $D1\times D2\times \cdots \times Dn$ is called a *relation*, and Di are called the *domains* of R. Let r be an element of R, then r is an n-tuple ($r1$, . . . , rn) where ri belongs to Di. It is convenient to introduce the notation $r[Di]=ri$ in terms of which the projection of R on Di is defined as $R[Di]=\{r[Di]\text{: } r \text{ in } R\}$ where it is understood that any duplicate values are eliminated. If one visualizes R as a table with its elements appearing as rows, then $R[Di]$ is just the column corresponding to Di.

We have found it convenient to distinguish the projection $R[D]$ from the domain D itself, i.e., to distinguish a column from the set of its possible values. To do this, we have introduced the term *attribute* to stand for $R[D]$. An attribute can be viewed as a function on R taking values in D and its alternative notation $\{r[Di]\text{: } r \text{ in } R\}$ makes this clear. This point of view is important in understanding the syntax of QUEL.

QUEL: A RELATIONAL QUERY LANGUAGE

QUEL is a calculus based language. Though closely modeled after the data-sublanguage ALPHA of Codd,[10] it has some significant differences. Among these are the following:

(a) Rather than relying on a host language for arithmetical operations, QUEL is closed under such operations.
(b) QUEL is free of all quantifiers.
(c) Aggregation operations such as SUM and MAX are treated with much greater generality.

An initial version of QUEL became operational in October, 1974 which used punctuation as delimiters.[7] The system implementors found this unnatural and the delimiters were changed to keywords. Since the parsing of QUEL is done by YACC, this modification was easily implemented. The designers of SEQUEL[11] are to be credited with emphasizing the desirability of keywords in relational languages. Our experience has confirmed that keywords are nearly universally preferred over alternative delimiters.

Each query of QUEL contains one or more Range-Statements and one or more Retrieve-Statements. We shall use { } to denote "one or more" and [ ] to denote "zero or more". With these conventions the form of a query in QUEL can be expressed as

Query
= {Range-Statement} {Retrieve-Statement}

* Research sponsored by the National Science Foundation Grant GK-43024x, U.S. Army Research Office—Durham Contract DAHCO4-74-G0087, the Naval Electronic Systems Command Contract N00039-71-C-0255, and a Grant from the Sloan Foundation.

Range-Statement

=RANGE of {Variable} IS Relation

Retrieve-Statement

=RETRIEVE INTO Result-name (Target-List)
WHERE Qualification

Target-List = {Result-Domain = Function}

The goal of a query is to create a new relation for each Retrieve-Statement. The relation so created is named by the "Result-Name" clause and the domains in that relation are named by the "Result-Domain" names given in the Target-List. In the frequent case where the Function is simply Variable. Domain-Name, the Result-Domain name may be omitted and is then taken to be the same as the Domain-Name in the Function. Also, if the "Result-Name" is TERMINAL then the result of the query is displayed on the user's terminal. To create the desired relation, first consider the product of the ranges of all variables which appear in the Target-List and the Qualification of the Retrieve-Statement. Each term in the Target-List is a function and the Qualification is a truth function, i.e., a function with values true or false, on the product space. The desired relation is created by evaluating the Target-List on the subset of the product space for which the Qualification is true, and eliminating duplicate tuples.

Example: CITY(CNAME,STATE,POPULATION,AREA)
"Find the population density of all cities in California with population greater than 50k"

```
RANGE OF C IS CITY
RETRIEVE INTO W(C.CNAME,
              DENSITY=C.POPULATION/C.AREA)
              WHERE C.STATE='California'
              AND C.POPULATION>50K
```

(note the default used for CNAME = C.CNAME and that the result of the query is a relation W(CNAME,DENSITY)).

It is clear from the above discussion that the basic quantities used in QUEL are functions on products of relations. The allowed functions can be exceedingly complex and fall into three categories: (a) Functions resulting from arithmetical combinations of attributes. (b) Set valued functions such as "the set of cities for each state". (c) Aggregate functions obtained by aggregating set functions, e.g., "total population of the cities of each state." The precise definition of the allowed classes of functions will be given recursively as follows: Consider a nested sequence of sublanguages of QUEL

QUEL0, QUEL1, . . . , QUELn, . . .

Let Ci denote the class of all functions and Qi the class of all qualifications allowed in QUELi. We first define $C0$ and $Q0$.

$C0$

(a) Any constant is in $C0$.
(b) Any attribute is in $C0$.
(c) If f and g are in $C0$ then $f+g$, $f-g$, $f*g$, f/g, $f**g$ and $\log(f)g$ are in $C0$.

(Note: The functions being combined need not have identical arguments. The resulting function is a function of the union of the variables.)

$Q0$

(a) An atomic formula in $Q0$ has the form $f(comp)g$, where *comp* is any of the comparison operators: $<$, $\leq$, $=$, $\neq$, and f and g are in $C0$.
(b) $Q0$ consists of all sentences made up of atomic formulas connected by the Boolean connectives: NOT, AND, OR.

Comment: A function in $C0$ will be referred to as an *attribute-function.* The value of an attribute function for a tuple depends only on the data contained in that tuple. This is not true for functions in Ci for $i>0$. A similar comment applies to $Q0$ as well.

We now proceed to define QUELn recursively. Suppose $X=(X1, X2, \ldots, Xm)$ are the declared tuple variables with range $R=R1\times R2\times\cdots\times Rm$. Let X.f and X.qual be respectively a function and a qualification allowed in QUEL $(n-1)$. We define SET(X.f WHERE X.qual) as the set of f-values obtained by evaluating f on the subset of R for which X.qual is true, ie.,

SET(X.f WHERE X.qual)

= {X.f: X is in R AND X.qual = true}

Example:

X	CNAME	STATE	POPU
r1	SF	CAL	1M
r2	NYC	NY	6M
r3	CHI	ILL	4M
r4	LA	CAL	3M

SET(X.POPU WHERE X.STATE = CAL) = {1M, 3M}

SET(X.POPU WHERE X.POPU > 3M) = {4M, 6M}

Comment: By definition a set contains no duplicate values. However, it is useful to define SET' as the collection obtained by retaining duplicates, for example,

SET'(X.STATE WHERE X.POPU < 4M) = {CAL, CAL}

The aggregation operators COUNT, SUM, AVG, MAX, MIN have an obvious meaning when they operate on sets. If AGG is any of these operators, we shall adopt the notation

AGG(X.f WHERE X.qual)

= AGG(SET(X.f WHERE X.qual))

and AGG' will denote AGG(SET').

We shall refer to quantities of the form AGG(X.f WHERE

X.qual) as *aggregates.* An aggregate depends on the data contained in the range R but does not vary as X varies. The appearance of X merely serves to indicate the range. In this way, it acts as a dummy variable not unlike that in a definite integral. To put it more precisely, denote a function in Cn by $F(X, R)$ to indicate the fact that in general it depends on both the tuple X and on R overall. Then, we can say that constants depend on neither X nor R, functions in $C0$ depend on X but not on R, aggregates depend on R but not X.

Now suppose that f and g are in $C(n-1)$ and qual is in $Q(n-1)$. Define

SET(X.f BY X.g WHERE X.qual)

as a set valued function of X such that it is constant on any set of X for which g is constant and on such a set it is given by

SET(X.f BY X.g WHERE X.qual) (X.g = alpha)

= SET(X.f WHERE (X.g = alpha) AND X.qual)

Example:

X	X.STATE	SET(X.CNAME BY X.STATE WHERE X.POPU < 5M)
r1	CAL	{SF, LA}
r2	NY	empty
r3	ILL	{CHI}
r4	CAL	{SF, LA}

The notation AGG(X.f BY X.g WHERE X.qual) is now self-explanatory, and so are the notations SET′(X.f BY X.g WHERE X.qual) and AGG′(X.f BY X.g WHERE X.qual).

Example:

X	X.STATE	MAX(X.POPU BY X. STATE WHERE X.POPU < 5M)
r1	CAL	3M
r2	NY	0
r3	ILL	4M
r4	CAL	3M

Note that AGG(X.f BY X.g WHERE X.qual), unlike aggregates, is a function of both X and R and will be called an *aggregate-function.* It is a function of X through X.g and only through X.g. Thus, the three appearances of X play a mixture of roles. This is an objectionable syntactic feature, which however cannot be repaired by using a dummy variable for the first and last term. AGG(X′.f BY X.g WHERE X′.qual) involves aggregation on the product of the ranges of X' and X and means something quite different from AGG(X.f BY X.g WHERE X.qual). Several possible solutions have been considered but rejected for one reason or another. In particular, if X is restricted to be a single variable, then its presence can be suppressed in the first and third term. For the time being we have chosen not to impose such a restriction.

Set functions of the form SET(X.f BY X.g WHERE X.qual) can be combined by union, intersection, and relative complement. We can define the class of set functions allowed in QUELn as follows:

Sn

(a) $S0$ contains all constant sets.
(b) Sn includes $S(n-1)$.
(c) If f and g are in $C(n-1)$ and qual is in $Q(n-1)$ then SET(X.f BY X.g WHERE X.qual) and SET(X.f WHERE X.qual) are in Sn as are SET′(X.f BY X.g WHERE X.qual) and SET′(X.f WHERE X.qual)
(d) Sn is closed under union, intersection and relative complement.

The classes Cn and Qn can now be defined as follows:

Cn

(a) Cn includes $C(n-1)$
(b) If s is in Sn then AGG(s) and AGG′(s) are in Cn.
(c) If f and g are in $C(n-1)$ and qual in $Q(n-1)$ then AGG(X.f BY X.g WHERE X.qual) and AGG′(X.f BY X.g WHERE X.qual) are in Cn.
(d) If f and g are in Cn, then $f+g$, $f-g$, $f*g$, f/g, $f**g$ and $\log(f)g$ are in Cn.

Qn

(a) Qn contains $Q(n-1)$
(b) If f and g are in Cn, then f(comp)g is in Qn, where comp is any of the operators $<$, $\leq$, $=$, $\neq$.
(c) If u and v are in Sn then u(set-comp)v is in Sn where set-comp is any of the set-comparison operators: inclusion, strict inclusion, equality, and inequality.
(d) If s is in Sn and alpha is a value, then (alpha belongs to s) is in Qn.
(e) Qn is closed under Boolean combinations.

Example:

SUPPLY(S#,P#,PRICE)

Query: Find those suppliers whose price for every part that he supplies is greater than the average price for that part.

```
RANGE OF S IS SUPPLY
RETRIEVE INTO W(S.S#)
WHERE COUNT(S.P# BY S.S# WHERE S.PRICE
                    >
          AVG'(S.PRICE BY S.P#))
=COUNT(S.P# BY S.S#)
```

Comments:

(a) It is clear that the Qualification of the Retrieve-Statement is in $Q2$.
(b) Instead of using COUNT, we could also have used the operator SET. In terms of processing efficiency, COUNT is preferrable.

UPDATE COMMANDS

In addition to RETRIEVE, QUEL permits three commands; REPLACE, DELETE, and APPEND, which are update operations. The syntax of the update statements is nearly identical to that of queries. Range statements have the same form and interpretation. The update statements have the same basic form as Retrieve-Statements:

```
Command Result-Name(Target-List) WHERE
  Qualification
```

For the APPEND Command, "Result-Name" must be the name of some existing relation, onto which qualifying tuples will be appended.

For the REPLACE (and DELETE) Command, "Result-Name" must be a tuple variable which, through the qualification, indentifies the tuples to be modified (or deleted).

The Target-List must contain explicitly (or by default) the existing Domain-Names for the relation being changed (the DELETE Command has no Target-List).

A few examples will indicate the usage of the Update-Commands. All of the examples will use the following relations and RANGE statements

```
EMP(NAME,SAL,BDATE,MGR)
NEWEMP(NAME,AGE,SALARY)
RANGE OF E IS EMP
RANGE OF N IS NEWEMP
```

Example:

All new employees over 35 are to work for SMITH

```
APPEND TO EMP(N.NAME,SAL=N.SALARY,
  BDATE=1975-N.AGE,MGR='SMITH') WHERE
  N.AGE>35
```

Example:

Give all employees a ten percent raise who work for Jones

```
REPLACE E(SAL=1.1*E.SALARY) WHERE
  MGR='JONES'
```

Note: The keywords BY and IS may be used interchangeably with '=' anywhere in INGRES to improve readability.

Example:

Remove all employees who are in the EMP relation from the NEWEMP relation.

```
DELETE N WHERE N.NAME=E.NAME
```

DECOMPOSITION

QUEL is obviously a very high level language and the success of system implementation depends critically on how the commands are processed. Palermo[12] and Rothnie[13] have considered the problem of query-processing in a relational data base system and have offered interesting suggestions on its solution. Neither, however, has provided an algorithm which is adequately general to process QUEL in its present form. Our approach is both more general and simpler. For the time being, we have opted for a uniform algorithm to deal with all queries rather than special strategies for special situations. How the algorithm can be tuned for each query is a problem that must be addressed at a later date.

Our overall strategy can be simply stated. Rather than compiling QUEL into a lower level language, we shall decompose an arbitrary QUEL-query into a series of one-variable QUEL-1 queries, at which point most of the difficult problems will have disappeared. Thus, for QUEL the "optimization" which is necessary for all high level languages lies nearly entirely in the decomposition. How "optimization" can be achieved is a problem still very far from a satisfactory solution. In the first version of our implementation, nothing more than some common sense rules of thumb are being used in selecting alternative paths at each stage of the decomposition.

Decomposition of QUEL queries is made difficult by an inconsistency in the language, viz., set-valued functions are allowed in the Qualification but not in the Target List. The role of set functions in the Qualification is to take the place of the universal quantifier. They are not allowed in the Target List because the result would be a relation with set-valued elements, i.e., an unnormalized relation. For example, the query

```
RANGE OF X IS CITY
RETRIEVE INTO W(X.STATE,CITIES-OF-STATE
=SET(X.CNAME BY X.STATE))
```

is an illegal QUEL query because it would result in an unnormalized relation

	STATE	CITIES-OF-STATE
W	CAL	{SF, LA}
	ILL	{CHI}
	NY	{NYC}

The only way of processing set functions which is consistent with the restrictions of QUEL is to substitute specific values one at a time for the conditioning function (i.e., the middle argument). For example, SET(X.CNAME BY X.STATE) becomes successively SET(X.CNAME WHERE X.STATE =CAL), SET(X.CNAME WHERE X.STATE=ILL), etc. This strategy, when carried out on a multivariable query, can result in a combinatorial explosion in complexity. At the present time the use of set functions in QUEL statements is permitted but discouraged. Fortunately, a query involving set functions can often be replaced by one without them. The following example illustrates this point.

Query: which states have only cities with population less than 4M?

```
RANGE OF X IS CITY
RETRIEVE INTO W(X.STATE)
  WHERE SET(X.CNAME BY X.STATE
  WHERE X.POPU<4M)
  =SET(X.CNAME BY X.STATE)
```

The Qualification in the above query can be replaced by

```
COUNT(X.CNAME BY X.STATE WHERE
  X.POPU<4M) =COUNT(X.CNAME BY X.STATE)
```

For the remainder of this section we shall outline a general decomposition algorithm for those queries in QUEL which do not contain set functions. Sets, however, will be allowed. The overall strategy has two parts: (a) A QUEL*n* query will be replaced by a series of QUEL$(n-1)$ queries and one-variable QUEL1 queries. (b) A multivariable QUEL0 query will be decomposed into a series of one-variable QUEL0 queries. Thus, repeated applications of the algorithm will decompose any QUEL*n* query into a series of one-variable queries in QUEL1 or QUEL0.

(a) QUELn→QUEL$(n-1)$

Consider a query involving one or more tuple variables $X=(X1, X2, \ldots, Xm)$ with range $R=R1\times R2\times\cdots\times Rm$. Let its Qualification be denoted by $Q(X)$. Suppose the query contains (either in its Target List or in its Qualification) an aggregate function AGG(X.f BY X.g WHERE X.qual) where f and g belong to $C(n-1)$ and qual belongs to $Q(\text{n}-1)$.

(i) RANGE OF Y IS R

RETRIEVE INTO TEMPO(A=Y.f,B=Y.g)
WHERE Y.qual

Comment: The purpose of this statement is twofold: to convert the multivariable range R into a single relation TEMPO and to delineate the tuples which satisfy qual.

(ii) RANGE OF Y1 IS TEMPO

RETRIEVE INTO TEMP1
(Y1.B,C=AGG(Y1.A BY Y1.B))

Comment: Thus far, we have created the portion of AGG(X.f BY X.g WHERE X.qual), for X.g belong to SET(Z.g WHERE Z.qual).

(iii) RANGE OF Y IS R

RANGE OF Z IS TEMPO
RETRIEVE INTO TEMP2(B=Y.g,C=AGG(ϕ))
WHERE Y.g does not belong to SET(Z.B)

Comment: AGG(ϕ) means AGG operating on an empty set. We shall adopt the following convention:

SUM(ϕ) =0=SUM′(ϕ)
COUNT(ϕ) =0=COUNT′(ϕ)
AVG(ϕ) =undefined=AVG′(ϕ)
(error if occurs)
MAX(ϕ) = $-\infty$ (i.e., the smallest possible value for that domain)
MIN(ϕ) = ∞ (i.e., the largest possible value for that domain)

(iv) RANGE OF Z2 IS TEMP2

APPEND TO TEMP1(Z2.B,Z2.C)

Comment: We have finally created the desired aggregate function

(v) In the original query we add RANGE OF Z IS TEMP1, add the clause "AND (X.g=Z.B)" to the Qualification and substitute Z.C. for AGG(X.f BY X.g WHERE X.qual).

Comment: (iii) and (iv) are omitted if either X.g or X.qual is absent in the aggregate function.

(b) Multivariable QUEL0→One-Variable QUEL0

Suppose that the Qualification is expanded in a conjunctive normal form so that it consists of clauses connected by AND with each clause containing atomic formulas or their negation connected by OR.

(0) Stop if already one-variable.

(i) For each variable, say *X1* with range *R1*, collect all the attributes which depend on *X1* and all the clauses in the Qualification which depend only on *X1*. Say *D1*, *D2*, . . . , *Dk* are the attributes, and the clauses put together yield *Q1*(*X1*). Issue the query.

RANGE OF X1 IS R1
RETRIEVE INTO R1′(X1.D1,X1.D2,. . . , X1.Dk) WHERE Q1(X1)

(ii) Replace the range $R1$ of $X1$ in the original query by $R1'$

Comment: The purpose of (i) and (ii) is to limit the range of each variable in the original query to as small a relation as possible by projecting and by enforcing the part of the Qualification which operates only on this variable.

(iii) Take the variable with the fewest tuples in its range and substitute in turn the actual values of its tuples. This reduces the number of variables by 1. After each substitution, repeat (0), (i), (ii) and (iii).

Comment: Step (iii) will be referred to as *tuple-substitution* and represents the most time consuming step in the overall algorithm. The choice of which variable to substitute for is critical. Our criterion of choosing the variable with the fewest values is by no means optimal in general.

This concludes the discussion of decomposition of queries. Update statements are transformed into queries followed (perhaps) by a sequence of insertions and deletions of tuples.[6]

ACCESS METHODS

As a result of the steps in the preceding section a sequence of one-variable QUEL0 and QUEL1 queries are generated. These queries can be executed directly (in the worst case by a sequential scan of the relation tuple by tuple). Often the relation will be stored in such a way that a complete scan is not required. Also secondary indices can be declared and are used if possible to limit the number of tuples examined. Currently there are five modes of relation storage and more can be added easily by implementing a common set of access calls (get next tuple, get unique tuple with equality on offered key, find starting point of a scan, etc.). These conventions are further discussed in Reference 14. Currently, a relation owner can decide both the relation storage and what secondary indices (if any) to construct. Soon both decisions will be done automatically by the system.

The current access methods are

(1) Unsorted Tables. This access method is supported for ease of entering relations into the system and is used for temporary relations (workspaces).

(2) Hashed Tables.[15,16] These are used when interactions almost always specify equality on a given domain or set of domains. A division algorithm is used, bucket-size is the page size used by UNIX and overflows are handled by chaining.

(3) Order Preserving Address Computation with a variable number of parameters. This access method is useful when scans over portions of a relation must be performed and the order preserving nature of the function can be used to limit the scan. The number of parameters is data dependent and ranges from none to the number of data pages used by the relation. With no parameters it resembles the computed functions of Rothnie[17] and Rivest.[18] With a maximum number of parameters it resembles VSAM.[19] This function and the choice of the number of parameters is discussed in Reference 14.

(4) & (5) Compressed Access. Access methods 2) and 3) can optionally (and transparently to higher level software) apply a compression scheme to each data page. Currently only front compression is used but the scheme will become more sophisticated in the future. These two access methods are useful when a large space saving will result in decreased I/O activity. Of course, the price paid is coding and decoding tuples at each access.

QUERY MODIFICATION

A high-level and nonprocedural language has benefits beyond its power and ease of use. For our purposes a major benefit will be the possibility of solving a number of system problems in a unified way, viz., by query modification. Here "query" is to be interpreted broadly to include the update commands as well. The specific problem areas to be addressed are: access control, integrity verification, and the support of "views." A suggestion along similar lines was made by Boyce and Chamberlin.[20] Since the details of our approach to these problems are being reported elsewhere,[21,22] we shall confine ourselves to a brief account of the basic ideas here.

Access control

Problem: We wish to provide a means whereby the access of each user to the database can be selectively restricted.

Solution: We define a pseudo QUEL command called RESTRICT which has a syntax nearly identical to that of RETRIEVE. The access control for each user is specified by a set of RESTRICT statements. For example, suppose EMPLOYEE(NAME,DEPT,SALARY,MANAGER) is a relation, and the restriction on SMITH is given by

```
RANGE OF E IS EMPLOYEE
RESTRICT ACCESS FOR 'SMITH' TO EMPLOYEE
  WHERE E.NAME='SMITH' OR E.MANAGER=
  'SMITH'
```

The interpretation here is that SMITH can retrieve only the data on himself and on anyone whom he manages. Different restrictions may be in force for the update commands, for example,

```
RANGE OF E IS EMPLOYEE
RESTRICT UPDATE FOR 'SMITH' TO EMPLOYEE
```

Execution: The Qualification of every RESTRICT statement for a given user is appended to the Qualification of every one of his interactions by conjunction (i.e., AND). For example, if SMITH issues the query

```
RANGE OF E IS EMPLOYEE
RETRIEVE INTO W(E.SALARY)
  WHERE E.NAME='JONES'
```

it is automatically executed as

```
RETRIEVE INTO W(E.SALARY)
  WHERE E.NAME='JONES'
  AND (E.NAME='SMITH' OR
  E.MANAGER='SMITH')
```

Comment: Queries involving aggregation are naturally more difficult to handle, but nevertheless can be handled in a similar way.[21]

Integrity assurance

Problem: We want to provide a means for maintaining certain constraints or consistency conditions on the data in the face of updates. For example, suppose we require that the data satisfy the constraints: "no employee can earn more than 21K" which may well be violated by the update operation "give everyone earning less than 20K a 10 percent raise".

Solution: Introduce a QUEL command, called INTEGRITY which expresses the constraint in the form of a QUEL qualification, e.g.,

```
RANGE OF F IS EMPLOYEE
INTEGRITY CONSTRAINT IS E.SALARY≤21K
```

Execution: Each update command which potentially violates the integrity constraint will be filtered by the INTEGRITY statement. This is done by appending qualification from the INTEGRITY Statement. The algorithms become somewhat complicated when the INTEGRITY Statement involves more than one tuple variable or aggregation.[6] Here, we confine ourselves to an illustrative example. Suppose that a command to "give everyone earning less than 20K a 10 percent raise" is issued. In QUEL this takes the form

```
RANGE OF E IS EMPLOYEE
REPLACE E(SALARY BY 1.1*E.SALARY)
  WHERE E.SALARY<20K
```

This statement upon modification becomes:

```
RANGE OF E IS EMPLOYEE
REPLACE E(SALARY BY 1.1*E.SALARY)
  WHERE E.SALARY<20K
  AND 1.1*E.SALARY≤21K
```

Views

Problem: We wish to accommodate queries referencing relations which do not exist but can be derived from existing relations. Due to reorganization of the database, existing programs may have obsolete "views" that need to be supported.

Solution: We introduce a QUEL command called DEFINE, which relates a "view" of the database to reality. For example: Suppose that at one time the database contained a relation EMPLOYEE(E#,NAME,DEPT,SALARY,SPOUSE,#CHILD). After reorganization, this relation becomes two relations. JOB(E#,DEPT,SAL) and FAMILY(E#,NAME,SPOUSE,#CHILD). The DEFINE statement is given by

```
RANGE OF J IS JOB
RANGE OF F IS FAMILY
DEFINE EMPLOYEE(J.E#,J.DEPT,SALARY=
  J.SAL,F.NAME,F.SPOUSE,F.#CHILD)
```

Execution: Any interaction referencing a relation which is the "Result" of a DEFINE statement is translated into an interaction referencing the relations declared in the RANGE of the DEFINE statement. For example, consider the statement

```
RANGE OF E IS EMPLOYEE
REPLACE E(SALARY BY 1.1*E.SALARY)
  WHERE E.NAME='JONES'
```

Upon modification, it becomes

```
RANGE OF J IS JOB
RANGE OF F IS FAMILY
REPLACE J(SAL BY 1.1*J.SAL)
  WHERE F.NAME='JONES'
  AND F.E#=J.E#
```

CONCLUSION

The first, and a very primitive, version of our system is now implemented. At the time of writing of this paper only QUEL0 queries are permitted. A complete, but relatively unclever, implementation for QUEL, is expected by April, 1975. Access control is the first of the query-modification features to be implemented, and is now working. Other query-modification features and strategies for greater efficiency in processing are on the more distant horizon.

ACKNOWLEDGMENT

Implementation of INGRES is being undertaken by Jim Ford, Peter Kreps, Nancy McDonald, Carol Williams, Karel Youssefi and Bill Zook. Their dedication and resourcefulness never cease to amaze us.

REFERENCES

1. Ritchie, E. and K. Thompson, "The UNIX Time-Sharing System," *CACM*, *17*, 1974, pp. 365-375.
2. Johnson, S. C., *YACC—Yet Another Compiler-Compiler*, Bell Telephone Laboratory, Murray Hill, N.J.
3. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *CACM*, *13* (1970), pp. 377-387.
4. Codd, E. F. and C. J. Date, "Interactive Support for Non-Programmers: the Relational and Network Approaches," *Proc. of the 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control*, Ann Arbor, Mich., May 1974.
5. Date, C. J. and E. F. Codd, "The Relational and Network Approach: Comparison of the Application Programming Inter-

faces," *Proc. of the 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control*, Ann Arbor, Mich., May 1974.

6. Stonebraker, M., *High Level Integrity Assurance in Relational Data Base Management Systems*, University of California, Electronics Research Laboratory, Memorandum ERL-M473, August 1974.
7. McDonald, N., M. Stonebraker, M. and E. Wong, *Preliminary Design of INGRES*, University of California, Electronics Research Laboratory, Memorandum ERL-435-436, April, 1974.
8. Stonebraker, M. and E. Wong, *INGRES—A Relational Data Base System*, University of California, Electronics Research Laboratory, Memorandum ERL-M477, November, 1974.
9. McDonald, N. and M. Stonebraker, *CUPID—The Friendly Query Language*, University of California, Electronics Research Laboratory, Memorandum ERL-M483, December, 1974.
10. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," *Proc. of the 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control*, San Diego, Calif., November 1971.
11. Boyce, R. and D. Chamberlin, "SEQUEL—A structured English query language," *Proc. of the 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control*, Ann Arbor, Michigan, May, 1974.
12. Palermo, E. P., "A Data Base Search Problem," *Proc. 4th International Symposium on Computers and Information Science*, Miami Beach, December 1972.
13. Rothnie, J. B., "A—Approach to Implementing a Relational Data Management Data Management System," *Proc. of the 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control*, Ann Arbor, Mich., May 1974.
14. Held, G. and M. Stonebraker, "Access Methods in the Relational Data Base Management System—INGRES," *Proceedings of ACM-PACIFIC-75*, San Francisco, Ca., April, 1975.
15. Morris, R., "Scatter Storage Techniques," *CACM*, *11*, 1968, pp. 35-38.
16. Lum, V., "General Performance Analysis of Key-to-Address Transformation Methods Using an Abstract File Concept," *CACM*, *16*, (1973,) pp. 603-612.
17. Rothnie, J. B., Jr. and T. Lozano, "Attribute Based File Organization in a Paged Memory Environment," *CACM*, *17*, 1974, pp. 63-69.
18. Rivest, R., *Analysis of Associative Retrieval Algorithms*, Institute de Recherche d'Informatique et d'Automatique (IRIA), Rapport de Recherche No. 54, February, 1974.
19. Keehn, D. and J. Lacy, "VSAM Data Set Design Parameters," *IBM Systems Journal*, Vol. 13, No. 3, 1974.
20. Boyce, R. F. and D. D. Chamberlin, "*Using a Structured English Query Language as a Data Definition Facility*," IBM Research Report RJ1318, December, 1973.
21. Stonebraker, M. and E. Wong, "Access Control in a Relational Data Base Management System by Query Modification," *Proc. 1974 ACM National Conference*, San Diego, Calif., November 1974.
22. Stonebraker, M., *Implementation of Views and Integrity Constraints in Relational Data Base Systems by Query Modification*, to be published.

Evaluating inter-entry retrieval expressions in a relational data base management system

by JAMES B. ROTHNIE, JR.
Department of Defense Computer Institute
Washington, D.C.

INTRODUCTION

Among the most important current concepts in data base technology is the relational model of data base management. The theory was introduced by Codd[1] in 1970 and has since been expanded in a large number of articles by Codd,[2,3,4] and other authors, for example.[5,6,7,8,9] The relational concept models the user view of a data base. As a user view it offers many applications significant advantages over more traditional approaches such as that proposed by the CODASYL Data Base Task Group.[10,11] Unfortunately some of the desirable properties of the relational model, and in particular the very important attribute called data independence, make the concept difficult to implement efficiently. This paper will describe an approach to solving one of the key implementation problems. The concepts discussed here have been embodied in an experimental system called DAMAS developed at the Massachusetts Institute of Technology. A complete discussion of DAMAS can be found in Reference 12.

A brief review of relational concepts

It will be assumed in this paper that the reader is familiar with the basic relational concepts but the terminology and notation will be very briefly reviewed.

In the relational user view a data base consists of a set of named relations. Each relation can be viewed as a simple table, a construct approximately equivalent to the more traditional idea of a file. With the tabular picture in mind, each row of a relation is called a tuple, where a tuple corresponds to a file entry or record in more familiar terminology. A relation is a set of tuples and, hence, no two tuples in a relation are identical. Each column is given an attribute name. A tuple contains one value (possibly null) for each attribute defined for the relation of which that tuple is a member. A relation called "employee", for example, can be defined having the attributes "name", "salary", and "department". A tuple of "employee" would provide a description of an individual in terms of those three attributes, for example, (John Jones, 12000, Sales).

The data structure in the relational model, then, is very simple. A data base consists essentially of a set of tables. The power of the relational model is derived from the mechanisms a user has available to extract information from this simple data structure. Codd has specified a language called DSL/ALPHA[2] which can be used to express operations on a relational data base. This language is somewhat awkward syntactically and other authors have defined more congenial versions based on this original, for example.[13] DSL/ALPHA, however, remains the most commonly known and we will use it to express retrieval problems in this paper. Two example expressions will adequately illustrate the relevant DSL/ALPHA constructs.

To extract from the "employee" relation the names and salaries of all employees earning more than $10,000 and place these names in a new working relation called W1 we would use the following expression:

```
RANGE EMPLOYEE E
GET W1 E. NAME, E. SALARY:
E. SALARY>10000
```

The phrase "E. NAME, E. SALARY" is called the target list and it identifies those attribute values in selected entries which will be included in the new relation. "E. SALARY>10000" is the qualification and it describes the condition which tuples must satisfy to be selected for the new relation. "RANGE EMPLOYEE E" is the range declaration. It defines E to be a tuple variable which ranges over the EMPLOYEE relation. The role of a tuple variable is essentially that of a bound variable in a set definition. For example, the following definition indicates which EMPLOYEE tuples would be selected for W1 and illustrates the role of E:

W1={E where E is a tuple of EMPLOYEE and the SALARY attribute of E is greater than 10000.} Retrieval expressions, like this one, which contain only a single tuple variable play an important role in the implementation method to be presented and they are termed primitive Boolean conditions (PBC). The real usefulness of the tuple variable concept, however, arises only in expressions where more than one tuple variable is required in the qualification. Consider, for example, the following problem.

Suppose we add to the EMPLOYEE relation a new attribute, MANAGER, which indicates each employee's manager. If a tuple has the NAME attribute John Jones and MANAGER attribute Jack Smith, then Jack Smith is

John Jones' manager. To create a relation W2 containing the names of all employees who earn more than their managers, we use the following expression:

```
RANGE EMPLOYEE E
RANGE EMPLOYEE M SOME
GET W2 E. NAME: E. MANAGER=M. NAME
    AND E. SALARY>M. SALARY
```

In the qualification, "E. MANAGER=M. NAME" establishes the required managee/manager relationship between E and M while "E. SALARY>M. SALARY" expresses the requested salary relationship. The range declarations indicate that both tuple variables E and M range over EMPLOYEE. The key word SOME in the declaration for M indicates that M is existentially quantified. That is, to select a tuple for E to include in W2 we need to establish the *existence* of a pair (E,M) which will make the qualification true. We need not draw any data from M; we need only establish its existence.

DSL/ALPHA permits the use of any number of tuple variables, each of which can be unquantified, existentially quantified or universally quantified. The language supports standard retrieval and maintenance operations and offers a number of other features. For the purposes of this paper, however, we will confine our discussion to retrieval (GET) expressions involving two tuple variables, one unquantified and one existentially quantified.

The implementation problem

A key goal of the relational model is to facilitate data independence in a data base management system. Generally a system can be said to be data independent if users and application programs are unaffected by changes in the physical storage of the data base and by certain types of changes in the user view.

The relational model encourages a system architecture possessing this characteristic by offering a high level user view which is quite divorced from storage structure and search algorithm concepts. This gives a well conceived relational system flexibility in electing a physical representation and search procedures, based on performance considerations, and in *changing* those choices as required. This is in contrast to systems, typified by the DBTG proposal, which depend upon user procedures to identify desired data objects. The greater level of detail in the DBTG user view restricts the system's opportunity to make or change representational choices. Substantial changes on the physical side will almost certainly involve changes to user decisions and hence will require user involvement.

The challenge to the implementor of a data independent system is to achieve efficient execution in the absence of the detailed user guidance available to DBTG-like systems. This is particularly a problem for relational systems since their powerful user languages permit users to request complex computations very simply. It is critical that the system be able to handle such requests efficiently or execution times are likely to exceed practical limits. The problem primarily arises for retrieval requests involving multiple tuple variables. To see this, note that for the most simple-minded implementation technique, exhaustive search, the execution time for handling a one tuple variable request is proportional to N, the number of tuples in the relation. The similar measure for a selection expression involving M tuple variables, each ranging over the N tuple relation, is N^M, the number of unique ordered collections of N tuples taken M at a time. This, of course, is a very crude bound and it is easy to construct algorithms which can improve upon it. However, this simple analysis does illustrate the danger of an exponential explosion in computation for this type of query, a danger which does not exist in a PBC. The remaining sections of this paper will be devoted to the description of a technique for handling two tuple variable queries, a technique which offers a far better performance than the brute force bound.

AN APPROACH TO IMPLEMENTATION

The system macro-organization

The algorithms we will be discussing for evaluating two entry variable retrieval expressions can be conceptually partitioned into two major modules. One, called the storage module, is responsible for the physical representation of relations and for responding to certain types of requests from the second module, called the multi-tuple variable module (MTVM). These requests take the form of subroutine calls and they require the storage module to perform certain operations on the stored relations, including:

- returning a sequence of tuples satisfying some qualification;
- determining the existence of a tuple satisfying some qualification;
- eliminating from further consideration in the current computation all those tuples which satisfy some qualification.

The key to the organization of this technique is in the nature of the qualifications which the tuples to be identified by the storage module must satisfy. In each case, this qualification is a PBC, i.e., a one tuple variable qualification. The storage module appears to the MTVM as a set of rather powerful primitives for operating on relations given a PBC to identify the tuples of interest. It is the responsibility of the MTVM to use these PBC handling primitives as a part of an algorithm which is capable of handling more than one tuple variable. The main thrust of this paper is to describe how the MTVM accomplishes this for two tuple variable expressions.

There are in fact a variety of techniques available for handling PBC's (e.g., inverted files,[14] multi-list files,[15] and multiple key hashing.[16]) Each of these methods offers advantages over the others in certain situations, so that a system with important claims to generality should not depend on a single one of these alternatives for all relations. To deal with this problem the DAMAS implementation was designed to operate with an unlimited number of storage modules, all with identical interfaces to the MTVM but each offering a different means of storing relations and handling PBC's. The organization is sketched in Figure 1.

In this paper we will be focusing attention on the MTVM; the internal structure of storage modules will not be further considered. We will view each storage module as a collection of PBC handling primitives. Even the details of these primitive calls will be suppressed here; the missing details can be found in References 12 and 17. Our approach in this paper will be to follow the operation of the MTVM on an example problem and to suggest the associated storage module operations as appropriate.

The example multi-tuple variable module operation

Figure 2 shows three example relations and a GET statement in DSL/ALPHA. The GET statement contains two tuple variables T1 and T2 which range over the pictured relations R1 and R2 respectively. T2 is existentially quantified. The result of processing the statement is R3 and it contains the A4 and A5 attribute values from tuples R1 which satisfy the rather complicated qualification. The reader should examine this qualification and understand its logic. A tuple T1 of relation R1 will be selected if:

1. Its own attributes satisfy the condition (T1.A1<3 OR T1.A3=1) and T1.A2>6

and

2. There exists a tuple T2 is R2 such that the pair of tuples (T1, T2) have attributes which satisfy the condition T1.A4≤T2.A3 and T2.A1=T1.A1

The details of this qualification are not important of themselves, but it is helpful to understand them in considering the operation of the MTVM in evaluating this expression.

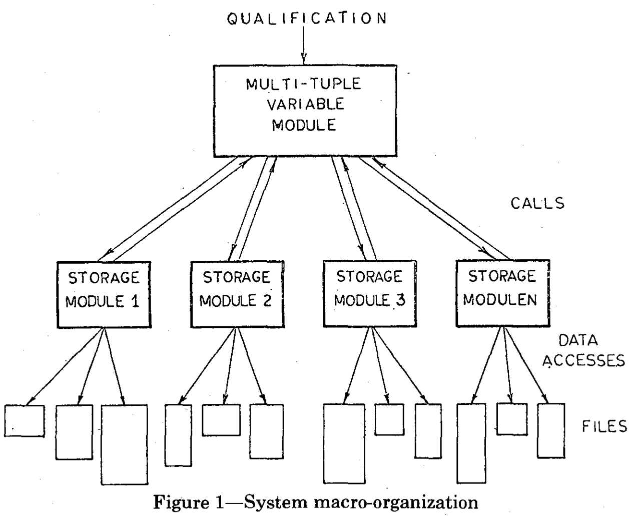


Figure 1—System macro-organization

Relation R1

	A1	A2	A3	A4	A5
(1)	1	7	1	3	1
(2)	1	7	2	4	1
(3)	1	8	2	4	3
(4)	1	6	2	2	2
(5)	2	2	2	3	1
(6)	2	8	2	4	2
(7)	2	7	1	6	2
(8)	2	7	1	7	2
(9)	2	8	1	6	2
(10)	2	8	1	7	2
(11)	3	3	1	3	1
(12)	3	4	1	2	1
(13)	3	6	2	1	2
(14)	3	8	1	3	1
(15)	4	1	2	5	2
(16)	4	8	2	4	2

Relation R2

	A1	A2	A3	A4	A5
(1)	3	5	4	3	7
(2)	3	6	3	1	2
(3)	2	1	2	1	5
(4)	1	1	5	2	1

Relation R3

	A4	A5
(1)	3	1
(2)	4	1
(3)	4	3

```
RANGE R1 T1
RANGE R2 T2 SOME
GET R3 T1.A4, T1.A5:
(T1.A1 < 3 OR T1.A3= 1)
AND (T1.A2 > 6 AND T1.A4 < T2.A3 AND T2.A1= T1.A1)
```

Figure 2—Example two-tuple variable selection

The mission of the MTVM is to evaluate the complete tuple selection expression. It accomplishes this by decomposing the expression into PBC's (one tuple variable qualifications) for evaluation by the storage modules. PBC's are derived from the original GET statement, possibly in combination with data retrieved from the relations during the course of the evaluation. The philosophy which guides the construction of PBC's and calls to the storage modules is an effort to derive as much information as possible from each access of the data base. Assuming that these accesses are the most expensive operations performed in the evaluation process, this is an attempt to gain maximum benefit for the cost.

The first action which the MTVM takes on encountering our example expression is to construct a PBC which eliminates from further consideration any tuple of R1 which can be discarded on the basis of its own attributes. For example, any tuple with attribute A1 greater than 2 and A3 not equal to 1 can be eliminated in this way. The MTVM implements this concept by constructing a PBC, called PBC1, which identifies that subset of R1 which *cannot* be eliminated in this way. The tuples satisfying PBC1 will then be subjected to further analysis. For this qualification, PBC1 is:

(T1.A1<3 OR T1.A3=1) and T1.A2>6

In general PBC1 can be constructed by setting all conditions involving T2 in the original qualification to TRUE and simplifying. The MTVM asks the storage module handling R1 to restrict its attention to the subset identified by PBC1 for all subsequent operations on the tuple variable.

Note that in the example relation this subset consists of the tuples identified by the numbers 1,2,3,6,7,8,9,10 and 14. Then the MTVM asks the module to return one tuple from this subset. Let this tuple be called E1.

Suppose that E1 is the tuple (1,7,1,3,1). The next task of the MTVM is to determine if this tuple satisfies the complete qualification. This involves ascertaining whether or not there exists a tuple in R2 which when paired with E1 satisfies the complete qualification.

To accomplish this the MTVM constructs a new PBC, called PBC2. PBC2 has the property that, if there exists a tuple in R2 for which PBC2 is true, then the tuple E1 satisfies the complete qualification. In general, PBC2 is derived by substituting the attribute values of E1 for the attributes associated with the tuple variable T1 and simplifying. For this particular qualification and choice of E1, PBC2 is:

$$3 \leq T2.A3 \text{ AND } T2.A1 = 1$$

As it happens, there is a tuple (1,1,5,2,1) in R2 which satisfies PBC2 so the target list from E1, (3,1), is included in the new relation R3.

At this point, it is possible to repeat the cycle by drawing a new value from the PBC1 subset, constructing a new PBC2, checking for qualified tuples in R2 and so forth. However, in this and many other cases, it is possible to gain additional benefits from the initial iteration. Specifically, given the existence of the tuple (1,1,5,2,1) in the relation R2, the MTVM can construct a PBC, called PBC3, which will identify additional tuples in R1 which fully satisfy the qualification because of the existence of this tuple in R2.

In general, PBC3 is constructed by replacing the T2 attributes in the original qualification with the contents of some tuple in R2 which satisfies PBC2, and simplifying. PBC3 in this case is:

$$(T1.A1 < 3 \text{ OR } T1.A3 = 1) \text{ AND}$$
$$T1.A2 > 6 \text{ AND } T1.A4 \leq 5 \text{ AND } 1 = T1.A1)$$

Since PBC3 is constructed from a known tuple of R2, any tuple of R1 which satisfies it will pass the complete qualification. In this instance when PBC3 is applied to R1, the additional tuples 2 and 3 are selected and their target lists are added to R3.

At this point, then, the MTVM returns to the storage module handling R1 and asks for another tuple E1 from the PBC1 subset. Suppose that this time E1 is (2,8,2,4,2). Again PBC2 is constructed, and for this tuple substitution we derive:

$$4 \leq T2.A3 \text{ AND } T2.A1 = 2$$

However, an examination of R2 will reveal that there is no tuple which satisfies this qualification. Hence E1 is not selected and its target list is not added to R3.

From here the MTVM can simply conclude the iteration and return for another tuple of the PBC1 subset. However, even the failure of a tuple to satisfy the qualification provides important information about the relations involved, and the system can take advantage of this information to avoid some later searching. Specifically, this failure indicates that there does not exist any tuple in R2 whose attributes satisfy the conditions in PBC2. Hence we can eliminate from further consideration any tuple in R1 whose attribute values will generate the same PBC2. Furthermore, if no tuple in R2 satisfies PBC2 then, clearly, no tuple exists which satisfies a qualification which is the same as PBC2 except that 4 is replaced by a larger value. An examination of the full qualification indicates that any tuple having a 4 or greater in attribute A4 and a 2 in attribute A2 will generate such a PBC2. The MTVM constructs a new PBC, called PBC4, to identify such tuples and it requests the storage module for R1 to eliminate these tuples from the PBC1 subset. In this case PBC4 is:

$$4 \leq T1.A4 \text{ AND } T1.A1 = 2$$

Tuples 7,8,9, and 10 are thereby discarded from the subsequent search.

After two complete iterations, then, we are left with only one tuple from the original PBC1 subset remaining to be considered. This is tuple #14 (3,8,1,3,1). However, this tuple need not be processed because its target list (3,1) is identical to a tuple already present in R3. Since a relation is a set it can have no two identical members and a second insertion of (3,1) will have no effect on the relation. Hence there is no reason to process the tuple. In order to recognize such situations, whenever a new tuple is added to R3 the MTVM can construct a PBC, called PBC5, which identifies all other tuples with identical target lists. The storage module for R1 is then asked to eliminate such tuples from the PBC1 subset. In this case, PBC5 is:

$$T2.A4 = 3 \text{ AND } T1.A5 = 1$$

If tuples satisfying PBC5 had been discarded when tuple #1 was selected for inclusion in R3, then at this point all tuples would have been handled and the algorithm complete.

An example performance analysis

This contrived example has illustrated the approach taken in the DAMAS implementation. While it was possible to construct a situation in which this specific algorithm was effective, it is equally possible to cite examples in which a simpler algorithm would be more effective. This is true because the use of the procedure steps associated with PBC3, PBC4 and PBC5 will not always identify enough tuples to justify their overhead. For this reason, each of these mechanisms is treated as an option to be employed or not depending on the situation.

Experimentation with DAMAS verified speculation that the decision to employ each option or not would have an important impact on performance and the optimal choices would vary from query to query. The following example from the experimental results will illustrate that point.

A relation called R1 was defined and loaded with 640 synthetically constructed tuples. The relation was assigned ten attributes named A1 through A10. Attribute values were randomly generated integers with all values, except those for A4, uniformly distributed from 1 through 16. The values for A4 were uniformly distributed integers ranging from 1 through 400. The storage module for R1 used a technique called multiple key hashing.[16] The details of this method are not relevant to an understanding of the example results. However, it is useful to know that the technique as applied here was most effective at identifying tuples with a given value for A1, somewhat less effective for A2 and A3, and still less effective for the other attributes.

The objective function used in comparing the various combinations of options available in DAMAS was the minimization of page accesses. The rationale for this choice was based on the observation that in the DAMAS computer system environment (MULTICS),[18] page accesses account for an overwhelming fraction of the total cost of querying a data base. Since a paged memory can serve as a gross model of any large data base environment, the results should be roughly extendable to other computer systems.

The first example query we will consider is the following:

```
RANGE R1 T1
RANGE R1 T2 SOME
GET W1 T1. all attributes: T1.A2=5 AND
  T1.A1≤T2.A5 AND T2.A2=5 AND T2.A3=5
```

When this retrieval expression was evaluated without using any of the special options, PBC3, PBC4 and PBC5, DAMAS performed 304 data page accesses. Of these, 248 occurred while searching R1 for a tuple satisfying PBC2—and failing to find one. A look at PBC2 will reveal generally the reason for this performance. PBC2 was of the form:

$$C \leq T2.A5 \text{ AND } T2.A2=5 \text{ AND } T2.A3=5$$

where C was replaced by the specific value drawn from attribute A1 of the tuple E1 used to generate PBC2. Because of the way that the storage module handled R1, a tuple satisfying this PBC might have occurred on eight different pages for any value of C. When there was no tuple in R1 which satisfied PBC2 the storage module searched all of these pages. As it happened there were 31 tuples in the PBC1 subset which failed to satisfy PBC2. Since each of these incurred eight accesses, the total of 248 for this portion of the algorithm resulted.

The relatively high cost of handling the case of negative results of PBC2 suggests that the use of PBC4 might have been fruitful. Recall that PBC4 provides a means of identifying some tuples which will fail PBC2 without performing a complete iteration for each of them. PBC4 in this case would be of the form:

$$T1.A1 \geq C$$

with C replaced by the A1 attribute of a tuple which failed on PBC2. As mentioned above, the storage module was effective in handling PBC's involving A1 so that processing of this PBC would not be excessively costly. Furthermore, the nature of this PBC suggests that a large fraction of the tuples which would fail PBC2 might be identified by processing PBC4.

When the experiment was actually performed, the first PBC4 constructed was:

$$T1.A1 \geq 6$$

Processing this expression required 12 page accesses and the result was the elimination of all remaining tuples which would fail PBC2. Hence at a cost of 12 page accesses required for PBC4, 248 page accesses were avoided. It might be argued that this substantial success was a fortunate accident. It is true that some other choices for E1 would produce less effective results. However, for this problem the highest cost possible for using PBC4 was 72 page accesses, still a substantial reduction from 248. In any case, the point that significant savings *can* be achieved with this option is clearly demonstrated.

The use of PBC3 in this case saved only 5 page accesses. This option was less effective than PBC4 because there were fewer tuples in the PBC1 subset which satisfied PBC2 and because the cost of handling each of them was less than the cost for those which fail PBC2. The use of PBC5 was clearly precluded because the target list contained all of the attributes of R1. Since relations are defined to contain no duplicates, searching for an identical target list was certain to be fruitless. If, however, the target list had been a single attribute, A1, then the use of PBC5 would have been valuable because of the high probability of identifying duplicates.

While PBC4 was effective in this example there are many instances in which it will be useless and, in fact, costly to employ. Consider, for example, the following retrieval expression:

```
RANGE R1 T1
RANGE R1 T2 SOME
GET W2 T1. all attributes:
T1.A3<3 AND (T1.A2=T2.A1 AND T2.A4<5)
OR (T1.A3=T2.A1 AND T2.A4>395)
```

For this problem PBC4 is of the form:

$$T1.A2=C1 \text{ AND } T1.A3=C2$$

where C1 and C2 are replaced by attribute values drawn from tuples which fail PBC2. The probability that a random tuple will satisfy PBC4 is, 1/256. Hence the use of PBC4 in this example is very unlikely to yield results which justify its overhead.

Strategy selection

The above performance analysis should clearly illustrate the following important point. For this approach to handling multi-tuple variable expressions to be effective, it is essential to employ an adequate method of deciding which options should be used in a particular situation. DAMAS did not include a facility for automatic strategy selection; users were asked to indicate which options they wished to employ. In the long run, however, it is clearly desirable, in the interest of data independence, user simplicity, and optimal choices, for the system to make this determination. The discussions in the last section which examined why the use of a certain option was effective in a given situation suggest a methodology for *predicting* which options will be hopeful. This section further explores that notion. A more detailed consideration of this topic can be found in Reference 19.

The MTVM selects a strategy by making a yes-no decision on the use of each of the special options PBC3, PBC4, and PBC5. Each of these decisions is based on an approximate calculation designed to indicate whether the costs avoided in using an option exceed the costs incurred in using it. The information concerning page accessing costs and numbers of tuples identified is obtained from the storage modules via special calls available to the MTVM. A sample decision on the use of PBC4 will illustrate the types of data obtained from the storage module and the calculations based on these data.

The MTVM requests the storage module to provide three items of information:

1) The expected number of tuples in the PBC1 subset which will be identified by one application of PBC4. This is obtained from the product of the probability that a random tuple will satisfy both PBC1 and PBC4 and the number of tuples in the relation. The call to the storage module includes PBC1 and PBC4 as arguments. The returned value is called N4.
2) The expected cost in page accesses of processing PBC2 when the result is false. Call this result C2.
3) The expected cost in page accesses of processing PBC4. Call this result C4.

Given these values the cost avoidance can be simply computed as N4*C2, the number of tuples identified times the cost of processing PBC2 for each one. If this value is greater than the cost incurred, C4, then PBC4 is used.

The same type of calculation can be performed for PBC3 and PBC5. Intuitively the approach seems likely to make reasonable choices in most situations but it has not been implemented and tested experimentally. Further exploration into the problems of strategy selection are clearly required.

SUMMARY

This paper has described via example a methodology for handling relational data base expressions involving two tuple variables. Somewhat in the manner of McDonald et al.,[9] this technique assumes the existence of one or more mechanisms for handling one tuple variable expressions (PBC's) and employs these as primitives in handling higher level expressions. The technique therefore is not concerned, at this level, with the detailed list manipulating operations by which the DIAM approach[20] handles inter-tuple associations. Such machinations might occur within the storage modules but they are invisible at the level of the MTVM where this paper was centered.

Unlike the McDonald approach, the technique described here does not involve a fixed sequence used in a largely identical way for all queries. Instead, this approach offers the possibility of using three options which in some situations can yield very substantial performance efficiencies. These options attempt to utilize the increasing information about the data base which is acquired with each access to a relation. Experimentation has verified that significant cost savings are possible when the appropriate combinations of these options are employed. An important difficulty, however, is the process of deciding which options to employ. A simple approach to automating these decisions was briefly discussed.

ACKNOWLEDGMENTS

The work was supported in part by the Cambridge Project under contract DAHC15 69 C 0347 of the Advanced Research Projects Agency and in part by the MIT Civil Engineering Systems Laboratory. Their support is gratefully acknowledged. Special thanks to Mr. John Rothnie for his able graphics assistance.

REFERENCES

1. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM13* No. 6, June 1970, pp. 377-387.
2. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," *Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control*, San Diego, November 1971.
3. Codd, E. F., "Further Normalization of the Data Base Relational Model," *Courant Computer Science Symposia 6*, "Data Base Systems", New York, May 1971, Prentice-Hall.
4. Codd, E. F., "Recent Investigations in Relational Data Base Systems," *Information Processing '74*, North Holland 1974.
5. Date, C. J. and P. Hopewell, "File Definition and Logical Data Independence," *Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control*, San Diego, November 1971.
6. Heath, I. J., "Unacceptable File Operations in Relational Data Base," *Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control*, San Diego, November 1971.
7. Palermo, F. P., "A Data Base Search Problem," *Fourth International Symposium on Computer and Information Science*, Miami Beach, December 1972.
8. Date, C. J. and E. F. Codd, "The Relational and Network Approaches: Comparison of the Application Programming Interfaces," *Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access, and Control*, Ann Arbor, May 1974.
9. McDonald, N., M. Stonebraker and E. Wong, *Preliminary Design of INGRES: Part I*, Electronics Research Lab. Univ. of Cal., Berkeley, ERL-M438, May 1974.

10. *CODASYL Data Base Task Group*—April 71 Report.
11. "CODASYL Data Description Language," *Journal of Development June 1973*, NBS Handbook 113.
12. Rothnie, J. B., *The Design of Generalized Data Management Systems*, Ph.D. Dissertation, Dept. of Civil Engineering, Mass. Institute of Technology, September 1972.
13. Chamberlin, D. D. and R. F. Boyce, "SEQUEL: A Structured English Query Language," *Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access, and Control*, Ann Arbor, May 1974.
14. Lefkovitz, D., *File Structures for On-line Systems*, Washington, D.C., Spartan Press, 1969, pp. 126-129.
15. Martin, L. D., "A Model for File Structure Determination for Large On-line Data Files," *File Organization, Selected Papers from File 68-Ah DAG Conference*, Amsterdam Swetsand Zeiglinger, New York (1969), pp. 223-245.
16. Rothnie, J. B. and T. Lozano, "Attribute Based File Organization in a Paged Memory Environment," *Comm. ACM 17* No. 2, February 1974, pp. 63-69.
17. Rothnie, J. B., "An Approach to Implementing a Relational Data Base Management System," *Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access, and Control*, Ann Arbor, May 1974.
18. Corbato, F. J. and V. A. Vyssotsky, "Introduction and Overview of the MULTICS System," *Proc. AFIPS 1965 FJCC*, Vol. 27, AFIPS Press, Montvale, N.J. pp. 185-196.
19. Rothnie, J. B., *The Design of Generalized Data Management Systems*, Ph.D. Dissertation, Dept. of Civil Engineering, Mass. Institute of Technology, September 1972, pp. 355-367.
20. Senko, M. E., E. B. Altman, M. M. Astrahan and P. L. Fehder, "Data Structures and Accessing in Data Base System," *IBM Systems Journal*, Vol. 12, No. 1 1973, pp. 30-93.

Views, authorization, and locking in a relational data base system

by D. D. CHAMBERLIN, J. N. GRAY and I. L. TRAIGER
IBM Research Laboratory
San Jose, California

INTRODUCTION

In the interest of brevity we assume that the reader is familiar with the notion of a relational data base. In particular, we assume a familiarity with the work of Codd[1,2,3,4,5] or Boyce and Chamberlin.[6,7,8] The examples in this paper will be drawn from a data base which describes a department store and consists of three relations:

```
EMP(NAME,SAL,MGR,DEPT)
SALES(DEPT,ITEM,VOL)
LOC(DEPT,FLOOR)
```

The EMP relation has a row for each employee, giving his name, salary, manager's name, and department. The SALES relation gives the dollar volume of each item sold by each department. The LOC relation gives the floor on which each department is located.

In References 6 and 7, Boyce and Chamberlin introduced SEQUEL, a data sublanguage based on English keywords and intended for interactive problem-solving by users who are not computer specialists. SEQUEL is a unified data definition and data manipulation language, based on the concept of a *mapping,* which allows users to select certain attributes from those rows of a table which satisfy some criterion. For example, the user may request the names and salaries of all employees in the shoe department:

```
SELECT NAME,SAL
FROM EMP
WHERE DEPT='shoe';
```

SEQUEL also allows attributes to be selected from two or more tables which have been joined together according to some stated criterion. For example:

```
SELECT NAME,FLOOR
FROM EMP,LOC
WHERE EMP.DEPT=LOC.DEPT;
```

produces a table of the names and floors of each employee by joining EMP and LOC on the DEPT column. (For a more complete treatment of joins, see References 4 and 6.)

This paper may be viewed as an extension of the ideas in Reference 7, which developed the concept of view and showed its applicability to extensible data structures, authorization, and integrity constraints, and Reference 15, which discussed the problems of locking relations and concluded that one must lock logical subsets of relations.

RELATIONS AND VIEWS

Defining relations

The SEQUEL system postulates a finite collection of *base relations.* The description of a relation includes a list of named *columns.* Each column (e.g., SAL) has the attributes *scope* (e.g., positive_integer), *comparability* (e.g., money/time), *units* (e.g., dollars/year), *representation* (e.g., DECIMAL (6)) and *role description* (e.g., "yearly compensation for services rendered").

The formal definition of EMP might be:

```
DEFINE EMP TABLE AS:
  NAME(SCOPE=ALPHA(*),DOMAIN=NAME,
  REPR=CHAR(*)),
  SAL(SCOPE=POS_INT,DOMAIN=MONEY,
  UNITS=DOLLARS,REPR=DEC(6)),
  MGR LIKE NAME,
  DEPT LIKE NAME EXCEPT (DOMAIN=
  DEPARTMENT),
  KEY=NAME,
  ORDER=ASCENDING NAME,
  INDEX NAME;
```

where the expressions to handle NAME, MONEY, POS_INT and DOLLARS have been previously defined.

Defining views

Simple variations of base relations may be obtained by:

(a) Renaming or permuting columns;
(b) Converting units or representation of a column;
(c) Selecting that subset of the rows of a relation which satisfy some predicate;
(d) Projecting out some columns of a relation

(e) Linking existing relations together into joins which can then be viewed as a single larger table.

Such variations can be obtained using the data definition facility. For example,

```
DEFINE ITALIAN_EMP VIEW AS:
  LIKE EMP EXCEPT (SAL.UNITS=LIRA,
  SAL.REP=DEC(9));
```

defines a view of employees paid in lira and expands the representation field appropriately. Thereafter, ITALIAN_EMP may be used as a relation. It may be placed anywhere in a SEQUEL statement that one could place the base relation EMP. All fetches from ITALIAN_EMP will have the salary field converted from dollars to lira. All stores into ITALIAN_EMP will store tuples into EMP with the salary field converted from lira to dollars.

To give a more sophisticated example, the table of employees and their locations is defined by:

```
DEFINE EMP_LOC VIEW AS:
  SELECT EMP,LOC
  WHERE EMP.DEPT=LOC.DEPT;
```

This statement defines the view:

```
EMP_LOC(NAME,SAL,MGR,DEPT,FLOOR).
```

Any SEQUEL query evaluates to a virtual relation which may be displayed on the user's screen, fed to a further query, deleted from an existing relation, inserted into an existing relation, or copied to form a new base relation. More importantly for this discussion, the query definition may be stored as a named *view*. The principal difference between a copy and a view is that updates to the original relations which produced the virtual relation will be reflected in a view but will not affect a copy. A view is a dynamic picture of a query, whereas a copy is a static picture.

There is a need for both views and copies. Someone wanting to record the monthly sales volume of each department might run the following transaction at the end of each month:

```
MONTHLY_VOLUME(DEPT,VOL)=
  SELECT DEPT,SUM(VOL) FROM SALES
  GROUPED BY DEPT;
```

The new base relation MONTHLY_VOLUME is defined to hold the answer, and its columns inherit the attributes of the SALES relation (e.g., the DEPT of MONTHLY_VOLUME inherits the scope, units, comparability, etc. of the DEPT in SALES). On the other hand, the current volume can be gotten by the view:

```
DEFINE CURRENT_VOLUME (DEPT,VOL)
VIEW AS:
  SELECT DEPT,SUM(VOL)
  FROM SALES GROUPED BY DEPT;
```

Thereafter, any updates to SALES will be reflected in the CURRENT_VOLUME view. Again, CURRENT_VOLUME may be used in the same ways base relations can be used. For example one can compute the difference between the current and monthly volume.

The semantics of views are quite simple. Views in SEQUEL can be supported by a process of substitution in the abstract syntax (parse tree) of the statement. Each time a view is mentioned, it is replaced by its definition. This fits well with the notion of nested mappings. Thereafter, the SEQUEL compiler and interpreter can treat views and nested mappings in a uniform way.

To summarize then, any query evaluates to a virtual relation. Naming this virtual relation makes it a view. Thereafter, this view can be used as a relation. This allows views to be defined as row and column subsets of relations, statistical summaries of relations and named joins. This mechanism contributes to:

(a) *Data independence:* giving programs a logical view of data, thereby isolating them from data reorganization.

(b) *Data isolation:* giving the program exactly that subset of the data it needs, thereby minimizing error propagation.

Views and update

Any view can support read operations; however, since only base relations are actually stored, only base relations can be physically updated. To make an update via a view, it must be possible to propagate the updates down to the underlying base relations.

If the view is very simple (e.g., ITALIAN_EMP above) then this propagation is straightforward. If the view is a one-to-one mapping of tuples in some base relation but some columns of the base are missing from the view, then update and delete present no problem but insert requires that the unspecified ("invisible") fields of the new tuples in the base relation be filled in with the "undefined" value. This may or may not be allowed by the integrity constraints on the base relation.

Beyond these very simple rules, propagation of updates from views to base relations becomes complicated, dangerous, and sometimes impossible.[5] Views derived from joins are not necessarily third normal form relations,[3] and hence may have unpleasant update properties. The types of updates which can be supported for various types of view will be discussed in a forthcoming paper. The following basic principles underlie our approach to the problem:

uniqueness rule: An insertion, deletion, or update to a view is permitted only if there is a *unique* operation which can be applied to the underlying base relations and which will result in *exactly* the specified changes to the user's view.

rectangle rule: An insertion, deletion, or update via a view must affect only information visible within the rectangle of the view.

These rules are illustrated by the following examples:

```
DEFINE MY_DEPT VIEW AS:
  SELECT EMP,LOC
  WHERE EMP.DEPT=LOC.DEPT
  AND EMP.MGR=USER;
```

where USER is a variable selected from the profile of the user of this program. This view is built from the join of the two base relations EMP and LOC. It allows one to see the name, salary, manager, department and floor of each employee who reports directly to the user of the view. If the user's name is Smith, it defines the rectangle: NAMExSALx('Smith')xDEPARTMENTxFLOOR which is a subset of the cartesian product which underlies the EMP_LOC relation defined previously. No actions using the MY_DEPT view can affect a tuple outside this rectangle. The SEQUEL statement:

```
DELETE MY_DEPT
WHERE SAL>15000;
```

would not delete all over-paid employees; it would only delete those overpaid employees who work for Smith. It really translates into the statement:

```
DELETE EMP
WHERE SAL>15000
AND MGR='Smith';
```

Since NAME is a key for EMP and DEPT is a key for LOC, MY_DEPT is a simple view which supports update, delete and insert. Of course, any tuple Smith updates via MY_EMP must have manager Smith before and after the update. Similarly, any tuple he inserts must have manager Smith, and he can only delete tuples with manager Smith. Each of these restrictions derive from the rectangle rule.

To give an example of the uniqueness rule, imagine that there is an employee who works in a department not listed in the LOC relation. For example, suppose the tuple (SCOTT,14000,SMITH,BOOK) appears in the EMP relation but that there is no book department in the LOC relation. Because of this, SCOTT will not appear in the join (in the virtual EMP_LOC relation defined previously) and so SCOTT will not appear in Smith's view MY_DEPT (which is a row subset of the EMP_LOC relation). Now if Smith inserts the tuple (FITZGERALD,13000,SMITH,BOOK,5) into his MY_DEPT view, this would propagate to inserting (FITZGERALD,13000,SMITH,BOOK) into EMP and (BOOK,5) into LOC. These inserts would add *both* Fitzgerald and Scott to Smith's view since they would add both to the join. This "side effect" is in violation of the uniqueness rule. Because of the possibility of such side effects, the MY_DEPT view cannot support insertions.

Another application of the uniqueness rule disallows support of insert or update to the CURRENT_VOLUME view defined previously, because there is not a unique way of propagating an updated SUM(VOL) to updates on the individual VOL entries in the base SALES relation.

AUTHORIZATION

If only one user has access to a data base, there seems little point in having any authorization mechanism beyond authentication on entry, although one still wants views for the reasons of conversion, isolation, etc., listed above. However, if several people expect to *selectively share* data then there must be some mechanism to protect and authorize access. Since one of the merits of a relational data base system is simplicity, we want a simple mechanism to dynamically create and share relations. This simplicity is important for a community of individuals who control their own data, as well as for a more centrally controlled system where authorization is handled by a (human) data base administrator. Following standard practice,[7,9,10,11] we use the view mechanism as the basis of the authorization mechanism (see Reference 12 or 13 for alternative approaches). The user has a catalog of named (base and virtual) relations. These give his only access to the data base. Each time a user defines a new base relation, a fully authorized view of it is placed in his catalog. The kinds of authorization we recognize are:

GRANT: the ability to grant this view to someone else or define a view on top of this view.

REVOKE: the ability to selectively reduce or revoke authorizations to this view.

DESTROY: the ability to destroy this view.

INSERT: the ability to insert into this view.

DELETE: the ability to delete tuples in the view.

And for each column of the view:

UPDATE: the ability to update values in this column.

We do not distinguish read access because read restriction can be gotten by eliminating columns from a view. All columns in a view are readable. Also, we do not distinguish "statistical" access or "manipulative" access. All known proposals for such access control are complicated to understand and easy to subvert. Owens[14] and Stonebraker and Wong[12] all present a convincing case against distinguishing statistical access. Our approach to statistical access is to use the view mechanism. For example, the CURRENT_VOLUME view described above gives only statistical access to the SALES relation in a very simple and understandable way.

Some fields (columns) within a tuple are more sensitive than others and therefore, update authorization is attached to the column of a view rather than to the entire view. Since relational operators distribute over a view, touching each tuple, it makes sense to authorize each visible tuple uniformly. For example, a manager may be authorized only to read the name and salary and to update the floor of any employee in the MY_DEPT view.

So each column of a relation or view has the attributes: scope, comparability, units, representation, role description, *and update authorization.* The view as a whole carries the authorizations for grant, revoke, destroy, insert, and delete.

Base relations when created have all fields updatable and are fully authorized for all operations. The creator may immediately define a view with non-updatable keys by (for example):

```
DEFINE EMPLOYEE VIEW AS:
  LIKE EMP EXCEPT (NAME.UPDATE='NO');
```

A derived view never has greater authorization than its parent view. If the view is not simple, then it automatically loses insert, delete, and update authorization. This is a good example of the interplay between authorization and views.

Granting and revoking authorization

Granting a view to Jones conceptually places a copy of the view definition into Jones' catalog of relations. Any user having grant authority to EMPLOYEE can grant it to another user with the same or reduced authority. For example:

```
GRANT EMPLOYEE TO JONES:
  (GRANT='NO',REVOKE='NO',DESTROY='NO');
```

This allows Jones to use EMPLOYEE, inserting in it, deleting from it, updating it, but prevents him from destroying the view or revoking it from someone else. Also it prevents him from granting the view to another or defining a view on top of EMPLOYEE. (Otherwise Jones could define an identical view and grant that view.) If Jones already has a relation named EMPLOYEE, the grant will fail.

Since Jones probably does not have the relation EMP in his catalog and since EMPLOYEE is defined in terms of EMP, the view must be interpreted in the context of the definer. On the other hand, the variable USER in the definition of MY_DEPT is local to the user of the view. Standard mechanisms are used to distinguish the definer's context from the user's context.

A second issue is revocation. When a view is destroyed, it is deleted from the catalog of all users to whom it was granted. This also invalidates all views which derive from that view. When anyone with revoke authority modifies the authorization of a view, that modification is again propagated to all views derived from that view. Further, anyone with revoke authority for the view may selectively revoke access to the view. For example:

```
    REVOKE EMPLOYEE FROM JONES;
```

revokes Jones' access to EMPLOYEE. One may imagine base relations and views organized into a hierarchy. If one view is defined in terms of another, then changes in the parent view will affect the child and *all* its descendants.

Checking authorization

When a transaction is "compiled" one may tell by the syntax of the statement which views are used by the transaction and for each view one can establish whether it is being granted, revoked, read, inserted into, deleted from or updated. We believe that much authorization will be *value dependent* and therefore must be checked at the time the transaction is run. For example, if a view is qualified by a selection criterion then each tuple which is inserted, deleted or updated must satisfy this criterion. For example all tuples entering and leaving MY_DEPT must be checked to see that the value of the MGR field is the name of the person running the transaction.

The entire SEQUEL system is carefully constructed so that mappings can be easily and uniformly composed. Once the update, insert, or delete is resolved to the underlying views and base relations, the translated tuples are tested against the selection criteria for the rectangles of those views. This process continues recursively until only base relations remain. If authorization, the uniqueness rule or the rectangle rule is compromised at any step, the operation faults. If a transaction tries to store outside its view, it is given a protection exception. If it tries to read outside its view, it is given the empty set as a response.

LOCKING

If several concurrent transactions access common data then there must be some protocol to synchronize their accesses. *This protocol should be invisible to the user.* The system is responsible for deciding what locks are required and whether they should be shared or exclusive locks (read or write access). Usually, a SEQUEL statement is the unit of consistency and locks are released at the end of a statement. To get consistency that spans multiple SEQUEL statements, the user may bracket the sequence of statements by the verbs: BEGIN_TRANSACTION and END_TRANSACTION. If two users each want to change the same data, one must wait for the other to finish. Under certain circumstances, one user may be forced to back up to the beginning of his transaction. If the transaction has not done any terminal input-output this is invisible to the user (except that the transaction takes a long time). If the transaction has done some I/O then backup will be automatic but visible. The issues of deadlock detection, preemption, and backup are resolved by the SEQUEL system using a priority-seniority scheduling scheme and a transaction log for backup.

In Reference 15 it is shown that if each transaction wants to see a consistent view of the data base, then locks must be held to the end of the transaction. It is further shown that the use of indices requires that transactions lock entire relations or that they lock *logical subsets* of relations.

To see that transactions must lock logical rather than physical subsets of a relation, imagine that Smith wanted to lock for read access all members of his MY_DEPT view. Scanning the EMP relation and locking all tuples with manager Smith would not prevent a new tuple with manager Smith from entering the EMP relation. For example, if Smith made a list of all his employees who make less than 15000 dollars and then made a list of all his employees who make less than 10000 dollars, the second set might not be a subset of the first! This problem of phantom tuples requires that Smith either lock the entire relation or that he lock the *logical subset* of tuples such that MGR='Smith'. This suggests the concept of predicate locks described as:

(RELATION,PREDICATE,((F1,A1),...(Fn,An)))

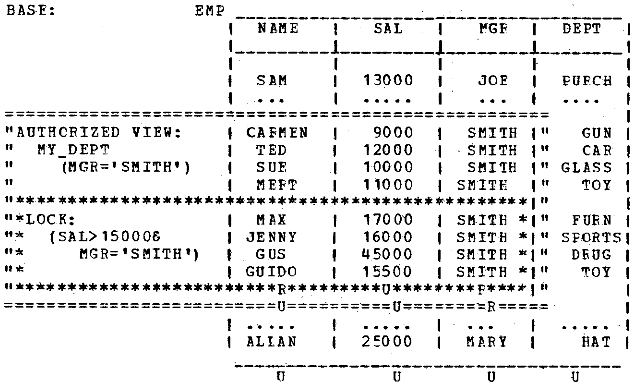
where PREDICATE is a selection criterion giving a row subset of the RELATION and the lock requests access of type Ai to field Fi. The kinds of access are read (shared) and write (exclusive). So for example,

(MY_DEPT,(MGR='SMITH'&SAL<10000),
((NAME,read),(MGR,read),(SAL,write))),

is a lock appropriate to the transaction:

UPDATE MY_DEPT SET SAL=1.10*SAL
WHERE SAL<10000;

which gives a 10 percent raise to each underpaid employee in Smith's view. The reason for specifying the kind of access to each column is to allow greater concurrency. Reference 15 contains a deeper discussion of the resolution of such locks. However their similarity to views should be obvious. Each view describes a rectangle of a (virtual or base) relation. Similarly, a predicate lock describes a rectangle of a relation and the access attributes of that rectangle. The view, authorization and lock mechanisms must each translate operations on a view to operations on base relations. Also, the view, authorization and the lock mechanisms each need to check that each tuple falls within the rectangles prescribed by the views and locks.



```
BASE:                    EMP ____________________________________________
                            | NAME     |   SAL    |    MGR   |   DEPT  |
                            |__________|__________|__________|_________|
                            |          |          |          |         |
                            |  SAM     |  13000   |   JOE    |  PURCH  |
                            |  ...     |  .....   |   ...    |  ....   |
=====================================================================  |
"AUTHORIZED VIEW:           | CARMEN   |    9000  |  SMITH   |"   GUN  |
"   MY_DEPT                 |  TED     |  12000   |  SMITH   |"   CAR  |
"     (MGR='SMITH')         |  SUE     |  10000   |  SMITH   |" GLASS  |
"                           |  MERT    |  11000   | SMITH    |"   TOY  |
"**********************************************************|"         |
"*LOCK:                     |  MAX     |  17000   | SMITH  * |"  FURN  |
"*   (SAL>15000&            | JENNY    |  16000   | SMITH  * |" SPORTS |
"*        MGR='SMITH')      |  GUS     |  45000   | SMITH  * |"  DRUG  |
"*                          | GUIDO    |  15500   | SMITH  * |"   TOY  |
"***************************R**********U**********R*****|"             |
============================U==========U==========R=====               |
                            | .....    |  .....   |   ...    |  .....  |
                            | ALIAN    |  25000   |  MARY    |    HAT  |
                             ------------------------------------------
                                 U          U          U         U
```

Figure 1—Base relation EMP, viewed from authorized view MY_DEPT and locked with respect to 'SAL>15000' in that view. The capital letters at the base of each rectangle give the authorization of that rectangle for that column

In most cases, locks will be finer than views (will be subrectangles of views) but in some complex cases the locks may extend to the entire base relation because the software is not smart enough to deduce the minimal lock predicate. Figure 1 illustrates the relationship between predicate locks, authorized views, and the base relation.

SUMMARY

Views prescribe what can be seen. *Authorization* prescribes what can be done to what is seen. *Locks* are a dynamic kind of authorization which prescribe what can be done to what is seen at this instant. Each of these concepts is an extension of its predecessor and all of them can be based on the concept of a defined relation with a qualifying predicate (a subrectangle of a virtual relation), where each column is tagged with read or write access and the whole view has authority qualifiers such as insert and delete.

STATUS OF IMPLEMENTATION

A single user SEQUEL system with SELECT, INSERT, DELETE, DEFINE, integrity constraints, and very sophisticated index selection has been operational since June 1974 at IBM's San Jose Research Laboratory. It is being experimented with at various locations within IBM. Current work is focused on a concurrent user system which will incorporate support for multiple views, locking, recovery and an advanced operating system interface.

ACKNOWLEDGMENTS

We have benefited from stimulating discussions with Ted Codd and Frank King on update propagation and with Kapali Eswaran and Raymond Lorie on the locking problem. Implementors of the SEQUEL system include: Morton Astrahan, Raymond Boyce, Don Chamberlin, Kapali Eswaran, Paul Fehder, Frank King, Raymond Lorie, and Jim Mehl.

REFERENCES

1. Codd, E. F., "A Relational Model for Large Shared Data Banks," *CACM,* Vol. 13, No. 6, June 1970, pp. 377-387.
2. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," *Proceedings of 1971 ACM SIGFIDET Workshop on Data Description, Access, and Control,* San Diego, California, November 1971.
3. Codd, E. F., "Further Normalization of the Data Base Relational Model," *Courant Computer Science Symposia,* Vol. 6, *Data Base Systems,* Prentice Hall, New York, May 1971.
4. Codd, E. F., "Relational Completeness of Data Base Sublanguages," *Courant Computer Science Symposia,* Vol. 6, *Data Base Systems,* Prentice Hall, New York, May 1971.
5. Codd, E. F., "Recent Investigations in Relational Data Base

Systems," *Proceedings of IFIP Congress 74,* Stockholm, Sweden, August 1974.

6. Boyce, R. F., D. D. Chamberlin, "A Structured English Query Language," *Proceedings of ACM SIGFIDET Workshop,* Ann Arbor, Michigan, May 1974.
7. Boyce, R. F., D. D. Chamberlin, *Using a Structured English Query Language as a Data Definition Facility,* IBM Research report: RJ 1318, San Jose, California, December 1973.
8. Boyce, R. F., D. D. Chamberlin, W. F. King III, M. M. Hammer, "Specifying Queries as Relational Expressions," *Proceedings of ACM SIGPLAN/SIGIR Interface Meeting on Programming Languages and Information Retrieval,* Gaithersburg, Md., November 1973.
9. Anonymous, *Information Management System Virtual Storage System (IMS/VS)—System/Application Design Guide,* IBM form No: SH20-9025, pp. 3.72–3.73, IBM Corporation, Palo Alto' California, 1971.
10. CODASYL, "Data Base task group report," ACM, New York, 1971.
11. Summers, R. C., C. D. Coleman, E. B. Fernandez, *A Programming Language Approach to Secure Data Bases,* IBM Los Angeles Scientific Center Technical Report: G320-2662, Los Angeles California, May 1974.
12. Stonebraker, M., E. Wong, *Access Control in a Relational Data Base Management System by Query Modification,* Electronics Research Laboratory memorandum: ERL-M438, UC Berkeley, California, May 1974.
13. Minsky, N., *Protection of Data-Bases, and the Process of Data-Base Interaction,* Computer Science Department, Rutgers, New Brunswick, N.J., September 1974.
14. Owens, R., *Primary Access Control in Large Scale Time-Shared Decision Systems,* Project MAC report TR-89, MIT, Cambridge, Mass., July 1971.
15. Eswaran, K. E., J. N. Gray, R. A. Lorie, I. L. Traiger, *The Notions of Consistency and Predicate Locks,* IBM Research report: RJ 1487, San Jose, California, December 1974.

Query by Example

by MOSHÉ M. ZLOOF
IBM T. J. Watson Research Center
Yorktown Heights, New York

INTRODUCTION

In the last few years we have witnessed a trend to appeal to the non-professional user who has little or virtually no computer or mathematical background.

The 'Query by Example' Language is an attempt in that direction. It operates on a relational Model of data as was introduced by Codd [1-5].

In this paper we deal only with normalized relations [1]. A relation is normalized if each of its domains is simple, i.e., no domain is itself a relation.

A normalized relation can be viewed as a table of *n* columns and a varying number of rows as illustrated in Figure I. Three properties of normalized relations are noteworthy to mention:

1. ALL rows of the table are distinct.
2. The ordering of the rows is immaterial.
3. The ordering of the columns is immaterial provided each has a distinct name.

EMP	NAME	SALARY	MANAGER	DEPARTMENT
	ANDERSON	8K	SMITH	TOY
	MORGAN	10K	LEE	COSMETICS
	•			
	•			
	•			

Figure 1—Employee relation

In THE LANGUAGE FACILITIES we introduce the concepts of the Language. ADDITIONAL EXAMPLES deals with additional examples. In GROUPING we introduce the concept of grouping. CONCLUSION deals with conclusions and remarks.

In addition a *sample data base* is available in the Appendix so that hopefully the user will refer to it in the course of learning the concepts of the Language.

THE LANGUAGE FACILITIES

In this section we introduce the Query by Example components. The concepts are described primarily through illustrations of queries and their answers, each illustration followed by a discussion to point out major features. The illustrations get progressively more complex until the whole scope of the Language is covered. In so doing, a user dealing with "simple" queries needs to study the system *only* to that point of complexity which is compatible with the level of sophistication required within the domain of those queries.

Furthermore, although the introduction of the concepts through illustrative examples reduces somewhat from the rigor of mathematical formulation through definitions, it is—in our opinion—more appealing to the casual user, which is one of the major aspects of Query by Example.

Most of the queries are drawn from the following tables (relations), which are part of a department store data base.

EMP (NAME, SAL, MGR, DEPT)
SALES (DEPT, ITEM)
SUPPLY (SUPPLIER, ITEM)
TYPE (ITEM, COLOR, SIZE)

—The EMP Table specifies the name, salary, manager and department of each employee.
—The SALES Table is a listing of the items sold by departments.
—The SUPPLY Table is a listing of the items supplied by suppliers.
—The TYPE Table describes each item by color and size.

At this point we are assuming that these tables are made available to the user upon calling them by name. In a subsequent paper, the creation, deletion, insertion and updating of these tables will be discussed in detail.

In this system the user basically formulates his query by *filling in* the appropriate table rows with an example of a possible answer. In fact for a large class of "simple" queries the user need only distinguish between the following two entities:

1. The 'example element' (variable) which must be underlined and
2. The 'constant element' which should not be underlined.[1]

In addition the function 'P.' stands for 'print'. The user inserts a 'P.' before any data he wishes to be outputted.

Examples:

Q1. Print the red items:

The user fills in the TYPE Table in the following manner.

TYPE	ITEM	COLOR	SIZE
	P.PEN	RED	

Since the query is concerned with red items, RED is a 'constant element' and is, therefore, not underlined. On the other hand, the underlined element PEN referred to as an 'example element' is entered as an example of a possible answer. Actually a pen may not necessarily be an element of the data base and can be substituted by DRESS, WATER or a variable X without altering the meaning of the query.

One of the reasons we are using an example element instead of a variable is that it gives us the freedom to use an entity which is partly variable and partly constant (see GROUPING). The SIZE Column can either remain blank or can be filled with an example element as well.

Considering the sample data base at the end of the paper the answer to this query is:

ITEM
LIPSTICK
PENCIL

For those users interested in the mathematical formulation of the queries, each query will be reformulated in predicate calculus

Q1. $\{x{:}\exists y(x, \text{RED}, y) \in \text{TYPE}\}$

Q2. What colors of ink are available?

TYPE	ITEM	COLOR	SIZE
	INK	P.BLACK	

In this case the 'P.' is in the color column since we want a listing of the colors of ink. BLACK is the example element.

ANS:

COLOR
GREEN
BLUE

Q2. $\{x{:}\exists y(\text{INK}, x, y) \in \text{TYPE}\}$

Q3. Find the department(s) that sells an item(s) supplied by the supplier Parker.

Here the user fills in both the SALES and the SUPPLY Tables as follows.

SALES	DEPT	ITEM
	P.TOY	ROD

SUPPLY	ITEM	SUPPLIER
	ROD	PARKER

ANS:

DEPT
HOUSEHOLD
TOY
STATIONARY
HARDWARE

Note: The example element ROD (linking variable) is included in both tables, implying if an item is sold by the department in question that *same* item has to be supplied by Parker. (Pretty much the same way one would scan the data base manually to find the answers.)

Q3. $\{x{:}\exists y((x, y) \in \text{SALES} \land (y, \text{PARKER}) \in \text{SUPPLY})\}$

Q4. Find the supplier(s) that supplies an item(s) sold by the TOY Department.

SALES	DEPT	ITEM
	TOY	PEN

SUPPLY	ITEM	SUPPLIER
	PEN	P.GM

ANS:

SUPPLIER
PARKER
BIC
REVLON

Q4. $\{x{:}\exists y((\text{TOY}, y) \in \text{SALES} \land (y, x) \in \text{SUPPLY})\}$

Q5. List the names, salaries, and managers of employees in the TOY Department.

EMP	NAME	SAL	MGR	DEPT
	P.JONES	P.10K	P.SMITH	TOY

ANS:

NAME	SAL	MGR
ANDERSON	6K	MURPHY
NELSON	6K	MURPHY
HENRY	9K	SMITH

Here the multiple output was achieved simply by inserting P. in the NAME, SAL, and MGR columns. The only constant element is TOY.

At this point we should mention that as long as an example element is not used for linkage purposes one can write just the function P. leaving blank space in place of the element. Thus in Q5. one can dispose of JONES, 10K, and SMITH.

Q5. $\{(x, y, z){:}(x, y, z, \text{TOY}) \in \text{EMP}\}$

The following additional types of operators and functions are part of the system:

numeric comparisons: $= \neq < \leq > \geq$
negation operator: $\neg$
the operators JOIN, ALL & ALL D. (explained later) and built in functions; SUM, COUNT, AVE, MAX, MIN, etc.

Q6. Pṛint out a list of all the departments, the items they sell and the suppliers that supply these items.

In this case we must first join the SALES Table with the SUPPLY Table on the common attribute ITEM and then apply the function P. as follows.

SALES	DEPT	ITEM
	TOY	PEN

SUPPLY	ITEM	SUPPLIER
	PEN	BIC

JOIN: SALES/SUPPLY	DEPT	ITEM	SUPPLIER
	P.TOY	P.PEN	P.BIC

The JOIN operator specifies joining the SALES and the SUPPLY Tables. The example element PEN appears in both tables to indicate a natural join on the common attribute ITEM.

ANS:

DEPT	ITEM	SUPPLIER
STATIONARY	DISH	DUPONT
HOUSEHOLD	PEN	PARKER
·	·	·
·	·	·
·	·	·
COSMETICS	LIPSTICK	REVLON
TOY	PEN	PARKER

Q6. $\{(x, y, z){:}\exists u (x, y) \in \text{SALES} \wedge (u, z) \in \text{SUPPLY} \wedge y = u\}$

Note: if the example element PEN does not appear in both tables, i.e., $y \neq u$, the join of these tables clearly becomes a Cartesian product.

Q7. Find the name(s) of any employee(s) who earns more than his (their) manager(s).

EMP	NAME	SAL	MGR	DEPT
	P.JONES	>10K	PETER	
	PETER	10K		

ANS:

NAME
LEWIS
HOFFMAN

If PETER is an example of such a manager and if PETER earns 10K (as an example) then JONES is an example of an employee who earns more than 10K (indicated by the > operator) and, therefore, more than his manager. It should be noted that the *order* of the rows is immaterial.

Q7. $\{x{:}\exists y \exists z \exists u \exists w \exists l \exists m ((x, y, z, u) \in \text{EMP} \wedge (z, w, l, m) \in \text{EMP} \wedge y > w)\}$

Q8. Find the department(s) that sells Pens *and* Pencils.

SALES	DEPT	ITEM
	P.TOY	PEN
	TOY	PENCIL

ANS:

DEPT
STATIONARY
TOY

Here, in order to account for the AND, the same example element TOY is used in both rows since the same department has to sell both items.

Q8. $\{x{:}(x, \text{PEN}) \in \text{DEPT} \wedge (x, \text{PENCIL}) \in \text{DEPT}\}$

Q9. Find the department(s) that sells Pens *or* Pencils.

SALES	DEPT	ITEM
	P.TOY	PEN
	P.HARDWARE	PENCIL

ANS:

DEPT
HOUSEHOLD
STATIONARY
TOY

Here two different example elements are used to account for the OR since a department that sells pens does not necessarily have to sell pencils.

Q9. $\{x{:}(x, \text{PEN}) \in \text{DEPT} \vee (x, \text{PENCIL}) \in \text{DEPT}\}$

Q10. Find the department(s) that sells *all* the items supplied by the supplier Parker.

SALES	DEPT	ITEM
	P.HOUSEHOLD	$\begin{bmatrix}\text{ALL }\underline{\text{PEN}}\\ \bullet\end{bmatrix}$

SUPPLY	ITEM	SUPPLIER
	ALL PEN	PARKER

ANS:

DEPT
STATIONARY
TOY

ALL PEN is defined to be the set of all the items supplied by Parker. The dot ('•') under ALL PEN in the SALES Table indicates that the department(s) in question *may* sell more than all the items supplied by Parker. On the other hand, if we wish to indicate that the department(s) in question has to strictly sell more than ALL PEN, it will be written as

$$\begin{bmatrix}\text{ALL }\underline{\text{PEN}}\\ \underline{\text{PENCIL}}\\ \bullet\end{bmatrix}\qquad (*)$$

The brackets in the ITEM column have no meaning other than grouping the dot with ALL PEN.

Q10. $\{x{:}\forall y((y,\ \text{PARKER})\in\text{SUPPLY}\rightarrow(x,\ y)\in\text{SALES})\}$

(*) Formally $\text{ALL }\underline{\text{PEN}} \subseteq \begin{bmatrix}\text{ALL }\underline{\text{PEN}}\\ \bullet\end{bmatrix}$

and $\text{ALL }\underline{\text{PEN}} \subset \begin{bmatrix}\text{ALL }\underline{\text{PEN}}\\ \underline{\text{PENCIL}}\\ \bullet\end{bmatrix}$

Q11. Find the department(s) such that *all* their items are supplied by Parker.

SALES	DEPT	ITEM
	P.HOUSEHOLD	ALL PEN

SUPPLY	ITEM	SUPPLIER
	$\begin{bmatrix}\text{ALL }\underline{\text{PEN}}\\ \bullet\end{bmatrix}$	PARKER

ANS:

DEPT
HARDWARE
TOY

Here the dot is under the ITEM column in the SUPPLY Table meaning that the Supplier Parker may supply more than all the items sold by the department in question.

Q11. $\{x{:}\forall y((x,\ y)\in\text{SALES}\rightarrow(y,\ \text{PARKER})\in\text{SUPPLY})\}$

Q12. Find the department(s) which sell *only all* the items supplied by Parker.

SALES	DEPT	ITEM
	P.HOUSEHOLD	ALL PEN

SUPPLY	ITEM	SUPPLIER
	ALL PEN	PARKER

ANS:

DEPT
TOY

Here the sets on both sides have to be equal thus there is no dot.

Q12. $\{x{:}\forall y((x,\ y)\in\text{SALES}\leftrightarrow(y,\ \text{PARKER})\in\text{SUPPLY})\}$

When a function such as P. SUM. COUNT. etc., precedes the operator ALL, the set ALL X (where X is any example element) retains its duplicate elements (*). This is necessary for the many instances when the duplicate elements are to be included in the count. This is illustrated in the next query.

Q13. Find the total salaries of the employees in the TOY Department.

EMP	NAME	SAL	MGR	DEPT
		P.SUM.ALL 10K		TOY

ANS:

SAL SUM =
21K

In this case the elements 6, 9, and the duplicate element 6 are summed. On the other hand if one wishes to exclude duplicate elements, the operator ALL D. is used where the 'D.' stands for differen‘ or distinct. This is again illustrated in the next example.

(*) Actually ALL X becomes a multi-set or a 'Bag' (in computer science terminology) where mapped duplicate elements are retained.

Q13. SUM $*\{x{:}\exists y\exists z(y,\ x,\ z,\ \text{TOY})\in\text{EMP}\}$

where the asterisk indicates that it is an operation on a multi-set.

Q14. How many colors of pencils are there?

TYPE	ITEM	COLOR	SIZE
	PENCIL	P.COUNT.ALL D.GREEN	

ANS:

COLOR COUNT =
2

(namely: red, blue)

Had we used the operators P.COUNT. ALL GREEN, the color blue would have been counted twice and the answer would have been '3'.

Q14. COUNT$\{x{:}\ y(\text{PENCIL},\ x,\ y)\in\text{TYPE}\}$

Q15. Among all departments with total salaries greater than 22K, find those departments which sell pens.

EMP	NAME	SAL	MGR	DEPT
		(SUM.ALL 10K) >22K		TOY

SALES	DEPT	ITEM
	P.TOY	PEN

ANS:

DEPT
STATIONARY

Q15. $\{x:(x, \text{PEN})\in\text{SALES} \wedge \text{SUM}$

$*\{y:\exists z\exists u(z, y, u, x)\in\text{EMP}\}>22k\}$

Note: Again the asterisk indicates the summation is performed over a multi-set.

Q16. Find item(s) that come in colors other than green.

TYPE	ITEM	COLOR	SIZE
	P. ROD	¬ GREEN	

ANS: The whole column of items except PEN will be printed. INK will be printed even though it comes in green, because it also comes in blue, thus satisfying the stipulation in the query.

Q16. $\{x:\exists y\exists z(y\neq\text{GREEN} \wedge (x, y, z)\in\text{TYPE})\}$

Q17. List all the items except the ones which come in green.

TYPE	ITEM	COLOR	SIZE
	ROD	GREEN	
	P. ¬ ROD		

ANS: The whole column of items except PEN and INK will be printed.

Unlike Q16, Q17 requires the elimination of any item that comes in green, even if the same item comes in other colors. In other words, the green items are sorted out and subtracted from the set of all items, leaving the complement set of non-green items. This complement set in our sample data base is the set of all the items except PEN and INK.

Q17. $\{x:\forall y\forall z((x, y, z)\in\text{TYPE}\rightarrow y\neq\text{GREEN})\}$

We must point out that if the data to satisfy the query are insufficient, the system prints 'NONE' in the appropriate column.

In addition, the system can be used as a verifier by completing the applicable columns with constant elements. If the element relations presented is positive, the system verifies that by printing the same constant elements. Otherwise, 'NONE' is printed in the column where the relation fails.

ADDITIONAL EXAMPLES

In this section we will formulate a collection of queries taken from various papers [5, 6, 7, 9] to illustrate major differences. No new features are introduced in this section.

Consider the following data base:

SUPPLY (SUPPLIER, PART NAME, JOB NAME)
PART (PART NAME, TYPE)
JOB (JOB NAME, LOCATION)

Q18. Find the names of suppliers who supply a job located in New York with all parts of Type A.

SUPPLY	SUPPLIER	PART NAME	JOB NAME
	P. ACME	[ALL. ROD]	BULB

PART	PART NAME	TYPE
	ALL ROD	A

JOB	JOB NAME	LOCATION
	BULB	NEW YORK

Consider the following data base:

EMP (NAME, SAL, MGR, DEPT)
SALES (DEPT, ITEM, VOL)
SUPPLY (COMP, DEPT, ITEM, VOL)
LOC (DEPT, FLOOR)
CLASS (ITEM, TYPE)

Q19. Find companies, each of which supplies every item of type A to some department on the second floor.

LOC	DEPT	FLOOR
	TOY	2

CLASS	ITEM	TYPE
	ALL PEN	A

SUPPLY	COMP	DEPT	ITEM	VOL
	P.PARKER	TOY	[ALL PEN .]	

Note: We can start formulating the query in any table, since the order is immaterial.

Consider the following data base:

EMP (MAN #, NAME, JOB CODE, SAL, DEPT #)
DEPT (DEPT #, NAME, MGR)

Q20. Find the information contained in the department record concerning departments having more than 20 employees whose job code is 802.

EMP	MAN #	NAME	JOB CODE	SAL	DEPT #
		(COUNT.ALL JIM)>20	802		30

DEPT	DEPT #	NAME	MGR
	P. 30	P.JONES	P. SMITH

GROUPING

In THE LANGUAGE FACILITIES we mentioned that the reason we chose to underline an element to make it a variable is to enable us to have an entity that is partly variable and partly constant.

Example: the number 560 is read 56X, and the name JIM is read JXY, where X and Y are variables.

This concept of creating a variable by underlining is extended to a second line to group equivalence classes of a set.

Example:

DEPT
780

where the second line indicates grouping by departments.

For ease of reference we change the EMP Table to EMP (NAME, DEPT).

Q21. Count the employees by departments.

EMP	NAME	DEPT
	P. COUNT. ALL JIM	P. TOY

ANS:

NAME COUNT=	DEPT
2	HOUSEHOLD
3	TOY
3	COSMETICS
2	STATIONARY

Q21. $$\{\text{COUNT}\{(x, y_1)\in \text{EMP}\},\ \text{COUNT}\{(x, y_2)\in \text{EMP}\}\ \ldots,\ \text{COUNT}\{(x, y_n)\in \text{EMP}\}:\{y_1 \ldots y_n\} = \text{DEPT}\ \wedge\ :i\neq j\rightarrow y_i\neq y_j\}$$

Q22. Count the employees by departments that have the same first letter on the left.

EMP	NAME	DEPT
	P. COUNT. ALL JIM	P. TOY

The answer is the same as in the case of Q21. However, if there are two departments with the same first letter, their employees will be counted together.

Q23. Count employees by departments and managers.

EMP	NAME	DEPT	MGR
	P. COUNT ALL JIM	P. TOY	P. SMITH

CONCLUSION

In this paper we presented the data access portion of the Query by Example Language. We conclude that the *unique* features of this language are as follows:

1. The user has the perception of manual table manipulation.
2. The user has a pre-established *frame of reference*, i.e., the tables.
3. The user can easily pre-identify the relations to be used, resulting in an early reduction in the scope of the data base.
4. As opposed to linear-type languages where the user is constrained to one degree of freedom, here the user has multi-degrees of freedom in that the sequence of filling in the tables and the rows within the tables is immaterial. This implies that given a data base the system does not constrain the user's thinking process in any way while he/she is formulating the query. Take Q7 as an example. If the user's thinking process wishes to first choose a manager and then compare his/her salary to the salary of his/her employees, the query would be the same whatever the row order is, thus the system is capable of capturing the different ways different users approach the problem.
5. The sequence of the following steps is also immaterial.

   a) filling in the constant elements,
   b) linking the variables,
   c) specifying the output by the P. function (projection), and
   d) grouping.

6. It follows from 4 and 5 that Query by Example allows the user to divide the query into decoupled segments, making it declarative and highly non-procedural. In contrast, most linear-type and other languages require the user to first specify the information to be outputted and then structure the query accordingly.
7. Due to the decoupling features inherent in Query by Example, it can handle rather complicated queries without relinquishing its simplicity. This is in contrast to other languages where a lengthy and complicated query has to be artificially divided into multiple steps and then taken one at a time.

REMARKS

1. "Relational Completeness" and arithmetic operations will be covered in subsequent papers.

2. Papers related to Query by Example are listed in the References 10 and 11.

ACKNOWLEDGMENTS

To Peter de Jong I am most grateful for his helpful suggestions and general encouragement throughout the development of this work.

For their helpful discussions, I wish to thank John Gould and John Thomas of the Behavioral Science Group who are currently testing the system on subjects through a display. Also, I wish to thank Ashok Chandra, Ron Fagin, Pat Goldberg, and Irving Wladawsky of IBM Research Yorktown for their useful discussions.

REFERENCES

1. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM*, Vol. 13, No. 6, June 1970, pp. 377-387.
2. Codd, E. F., "Further Normalization of the Data Base Relational Model," *Courant Computer Science Symposia*, Vol. 6, *Data Base Systems*, Prentice-Hall, New York, May 1971.
3. Codd, E. F., "Relational Completeness of Data Base Sublanguages," *Courant Computer Science Symposia*, Vol. 6, *Data Base Systems*, Prentice-Hall, New York, May 1971.
4. Codd, E. F., "Normalized Data Base Structure: A Brief Tutorial," *Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access and Control*, San Diego, November 1971.
5. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," *Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access and Control*, San Diego, November 1971.
6. Boyce, R. F., D. D. Chamberlin, W. F. King III, and M. M. Hammer, "Specifying Queries as Relational Expressions," *Proceedings of ACM SIGPLAN/SIGIR Interface Meeting on Programming Languages and Information Retrieval*, Gaithersburg, Maryland, November 1973.
7. Astrahan, M. M., E. B. Altman, P. L. Fehder and M. F. Senko, "Concepts of a Data Independent Accessing Model," *Proc. 1972 ACM SIGFIDET Conference*, Denver, Colorado, November 29-30, 1972.
8. *Interactive Query Facility (IFQ) for IMS/360*, Publication No. GH 20-1074, IBM Corporation, White Plains, New York.
9. Chamberlin, D. D., and R. F. Boyce, *SEQUEL: A Structured English Query Language*, IBM Report, No. RJ1394.
10. Zloof, M. M., *Query by Example: The Invocation and Definition of Tables and Forms*, IBM Research Report, No. RC5115, February 1975.
11. Zloof, M. M., *Query by Example: Operations on the Transitive Closure*, IBM Report in preparation.

APPENDIX

SAMPLE DATA BASE

EMP	NAME	SALARY	MGR	DEPT
	JONES	8K	SMITH	HOUSEHOLD
	ANDERSON	6K	MURPHY	TOY
	MORGAN	10K	LEE	COSMETICS
	LEWIS	12K	LONG	STATIONARY
	NELSON	6K	MURPHY	TOY
	HOFFMAN	16K	MORGAN	COSMETICS
	LONG	7K	MORGAN	COSMETICS
	MURPHY	8K	SMITH	HOUSEHOLD
	SMITH	12K	HOFFMAN	STATIONARY
	HENRY	9K	SMITH	TOY

SALES	DEPARTMENT	ITEM
	STATIONARY	DISH
	HOUSEHOLD	PEN
	STATIONARY	PENCIL
	COSMETICS	LIPSTICK
	TOY	PEN
	TOY	PENCIL
	TOY	INK
	COSMETICS	PERFUME
	STATIONARY	INK
	HOUSEHOLD	DISH
	STATIONARY	PEN
	HARDWARE	INK

SUPPLY	ITEM	SUPPLIER
	PEN	PARKER
	PENCIL	BIC
	INK	PARKER
	PERFUME	REVLON
	INK	BIC
	DISH	DUPONT
	LIPSTICK	REVLON
	DISH	BIC
	PEN	REVLON
	PENCIL	PARKER

TYPE	ITEM	COLOR	SIZE
	DISH	WHITE	M
	LIPSTICK	RED	L
	PERFUME	WHITE	L
	PEN	GREEN	S
	PENCIL	BLUE	M
	INK	GREEN	L
	INK	BLUE	S
	PENCIL	RED	L
	PENCIL	BLUE	L

A psychological study of query by example

by JOHN C. THOMAS and JOHN D. GOULD
IBM T. J. Watson Research Center
Yorktown Heights, New York

INTRODUCTION

Many different query systems have been proposed.[1] One way to partition extant and hypothetical query systems is on the basis of how English-like they are. One approach for an easy-to-use query system is to allow the user to state a question in natural English. The system may then disambiguate the possible interpretations of this question on the basis of context[2] or on the basis of feedback questions to the user.[3] A second approach is to require the user to state his question in a formal language system, but one that uses an English-like grammar and vocabulary. IBM's Interactive Query Facility[4] (IQF) and SEQUEL[5] exemplify this type of approach. A third approach is to require the user to state his question in a formal language system that does not attempt to appear "English-like." Zloof's Query By Example language[6] is best described by this third approach. In this paper we demonstrate experimentally the ease and accuracy with which nonprogrammers learned and used this powerful Query By Example language.

It may seem that the first approach mentioned above, viz., allowing the user to state his question in natural English, must necessarily be the best approach and the one we should study. There are several reasons why this is not so. First, any natural English system in the foreseeable future is likely to place serious restrictions on the vocabulary and syntax of allowable, or at least useful, user inputs to the system. It may be difficult for the user to keep in mind such restrictions since daily conversation may tend to provide strong interference with the rules of such a system, and hence produce considerable forgetting. In this experiment, we provide evidence that Query By Example may be robust against such forgetting. Second, as a user becomes more familiar with a system, particularly if his job involves some percentage of fairly routine question asking, lengthy feedback dialogues may come to be perceived as a waste of time. Third, the need for this dialogue will increase the cost of the computer system. A fourth factor relates to data representation. Should the system represent data relations in natural English or in a more formal representation? Although it might be assumed that natural English would be the optimal way to represent data to users, studies of problem solving provide evidence that sentential representations are often non-optimal for humans.[7,8,9] Providing the user with a formal representation, such as Query By Example does with tables that implicitly supply relations among the data, may better help the user formulate and solve his problem.

Our primary purpose was to conduct an exploratory evaluation of a powerful query language that seemed promising for non-programmers to learn and use. Of specific interest were the time to learn the language, and the time, accuracy, and subjects' confidence in translating test questions stated in English into Query by Example. These test questions were selected from disparate regions of the space of potential questions, making sure to include some that required each of the features of the language. This was done to determine the factors that made this translation process difficult. We included some questions that involved concepts that previous work had shown are difficult for most people (e.g., conjunction vs. disjunction constructions and universal quantification)[10,11] and some questions studied with other query languages.[12]

METHOD

Subjects

Subjects were run in four successive groups; these consisted of four college students, 11 college students and recent graduates, and two groups of 12 high school students. Data on class standing and IQ were available for 23 of the 24 high school students. The mean class standing was 44/197 and the median IQ[13] was 115. There were 15 males and 24 females. The subjects ranged in age from 16 to 24 years. Four of the college students had some minor programming in school courses. Aside from this, none of the subjects had any experience in using computers.

Design and procedure

Subjects received about one hour and 45 minutes training on the major features of Query By Example. They then wrote translations of 20 test questions; this took about 40 minutes. After a short break, subjects received another 70 minutes of instruction, followed by another 20-question test, which took about an hour. The second test contained some questions designed to assess understanding of the concepts presented during the second training period, and it also contained some questions similar to

TABLE I—Sample Questions inQuery by Example

NAME	SALARY	AGE	MANAGER	DEPARTMENT
p. *Jones*	p. *23K*			Sports
p. *Smith* Riley				*Toys* *Toys*
p. *Scholz*				*Computers*

SUPPLY FILE

COMPANY	ITEM	# RECEIVED
IBM	*360*	

SALES FILE

DEPARTMENT	ITEM	# SOLD
Computers	*360*	

SUPPLIER DATA FILE

COMPANY	LOCATION	SIZE	PRESIDENT
IBM	Mass.		

those on the first test. The purpose of these latter questions was to measure the effect of adding other features (for example, set inclusion) to a subject's understanding of the "easier" part of the language. Two weeks after initial learning, six college students were available for retest. These students were given a test of 20 questions without any retraining. They were then given an hour refresher and given another test of 20 questions.

During testing, subjects were not given feedback about the correctness of their answers. If they had been given such feedback after writing each query, they would undoubtedly have written a higher percentage of queries correctly. They also could not refer to instructional material or notes used in training. However, subjects were provided a list of abbreviations and operators in the front of the test booklet. The order of questions within a test was rotated in five different sequences among subjects so that fatigue or practice effects would not systematically affect the estimates of question difficulty.

Query by example language

The Query By Example language that subjects were taught is described in detail elsewhere.[6] In this experiment subjects were given empty tables, containing only the table names and column names. Subjects wrote a query by filling in some of the columns in one or more of the tables with examples (variables), with constants, and with operators. Table I illustrates some of the features of the language.

The first query in the table represents the question "Print the names and salaries of the people in the sports department." The "p." in the columns labelled NAME and SALARY shows that those are the items to be printed out. "Jones" and "23K" are examples. They are underlined because they are examples. "Sports" is not underlined because it is a constant.

The second query in Table I represents the question "Print out the names of those people who work in the same department as Riley." "Toys" is underlined to indicate that it is a variable, in this case a linking variable. It links "Riley" to the people that will be printed out. These two lines could be interchanged.

The last line in the top table combined with the bottom three tables is the query for "Who are the people who work in a department that sells items supplied by companies located in Massachusetts." This problem requires the use of four tables, and it illustrates how tables are linked together. Note that Massachusetts is the only constant in this question; the other words are examples (variables) and hence are underlined.

Training

Subjects were trained in groups. Training was based primarily upon a lecture during which subjects were shown, via an overhead projector, a series of examples of about 100 queries written in Query By Example. Each example was explained in the lecture, and subjects were required throughout training to code many English questions into Query By Example. To do this, they were given forms that contained outlines of the data tables on which the training queries were based. Besides the lecturer, there was a second instructor present, and throughout training both instructors continuously checked on the accuracy of the students' queries. In this way both the students and the

TABLE II—Example Queries Used in Evaluation

1. Print the names of employees whose salary is less than $12,000, are over 28 years old, and are managed by White.
2. I think there must be about 30 people managed by White, who make $12,000. So . . . list the people over 28 who are managed by White. Of course, I'm only talking about those who make less than $12,000.
3. I'm thinking about a raise for someone from White's group. I want only older people who are underpaid. For starters, find the people who work for White, are over 28, and make less than $12,000.
4. Who else works in the same department as Riley?
5. List the names of employees who are younger than Anders' manager.
6. List people who work in departments that sell at least one item supplied by a company located in Massachusetts.
7. Print the names of anyone who makes more than Anders' manager and is younger than Smith's manager.
8. The accounting people need to know how many married women over 30, with no dependents, work in Dept. 300; and how many single employees who have at most a Junior College degree have worked for us for more than 5 years and are employed in St. Louis, in Dept. 400.

teachers had feedback while each concept of the language was being taught.

Task

Following training, each subject's task was to translate 40 questions stated in English into Query By Example. Table II provides some examples. These questions varied in several ways. Most were stated in a straightforward way. In some cases, formally identical problems were also stated in both a poorly expressed manner and in a way which included a rationale for asking the question. The first three examples provide examples of these variations. Queries also varied in the number of linking variables required: 0, 1, 2, 3, or 4. Sometimes these links were within a single table and sometimes between tables. Examples of queries requiring one linking variable within a table, two linking variables within a table, three linking variables among tables, and four linking variables within a single table are given as examples 4 through 7 in Table II. Questions also varied in the number of conjunctive constraints, the number of disjunctive sets, and in the number of operators used to compute quantities from a subset. Example 8 in Table II illustrates a questions with two disjunctive "sets", with each set requiring five conjunctive constraints. In addition, the question asks for an operation of counting to be performed.

Subjects were also required to write the time that they started reading each English question and the time that they completed writing each query. In addition, 35 subjects gave a confidence rating between 1 and 5 to indicate how sure they were that each query was correct. A "1" corresponded to "very sure correct", a "2" corresponded to "fairly sure correct", a "3" corresponded to "50-50 chance", a "4" corresponded to "fairly sure incorrect", and a "5" corresponded to "very sure it's incorrect."

Error analyses

The correctness of the queries that the subjects wrote was assessed by two people familiar with the syntax of the Query By Example system.

RESULTS

Overall

Mean training times varied between two hours and two hours and 55 minutes for the four groups. The proportion of correct queries was 0.67; mean time to complete a query was 1.6 minutes overall (s.d.$=$1.09); mean confidence rating was 1.8 (s.d.$=$0.82).

Individual differences

All subjects were able to learn the language and complete the experiment. The performance of the college students and high school students was nearly identical on the above four measures, and there also were no sex differences. The range of individual differences was 0.33 to 0.93 on proportion of queries correct ($F(38,1482)=5.24;p<0.001$); and 1 to 2.3 on confidence ratings ($F(34,1326)=12.12$);$p<0.001$). Thirty-four of the 39 subjects were correct on over 50 percent of the questions. Subjects who were confident also were relatively accurate ($r=-0.63;p<0.001$). Subjects who were relatively fast in writing queries were not significantly more confident ($r=0.23;p>0.05$) or more accurate ($r=-0.19;p>0.05$). The correlation between Otis[13] IQ and accuracy on the first 20-question test was 0.60 ($p<0.01$) and 0.50 ($p<0.05$) on the second 20-question test.

Problems

The proportion of subjects writing correct queries for each of the 40 questions varied from 0.26 to 1.0 ($F(39,1482)=12.59;p<0.001$). Twenty-six of the 40 questions were done correctly by the majority of subjects. The mean time to write queries varied from 0.83 minutes for the fastest problem to 3.64 minutes for the slowest problem ($F(39,1482)=26.12;p<0.001$). The mean confidence ratings varied from 1.3 to 2.5 ($F(39,1326)=10.24;p<0.001$). The correlation on the first 20-question test between mean accuracy and mean time to write a query for that question was -0.70 ($p<0.001$). On that same test, problems on which people were accurate were problems which they were also confident about

TABLE III—Proportion of Queries Correct for Various Confidence Ratings

	CONFIDENCE RATINGS				
	Very Sure Correct	Fairly Sure Correct	50-50 Chance	Fairly Sure Incorrect	Very Sure Incorrect
Proportion Correct	0.83	0.62	0.44	0.17	0.20
Total N	581	585	187	36	10

($r = -0.92; p < 0.001$); problems which they did quickly also were associated with high confidence ratings ($r = 0.78; p < 0.001$). These correlations were significant ($p < 0.01$) but lower on the second 20-question test.

Table III shows the probability of a problem being correct, given a particular confidence rating. As is clear from the table, a computer system could make a useful prediction about the correctness of a particular query by asking a user how likely he thinks it is that the query is correct.

Accuracy predictions

Three types of measures were used to predict the proportion of correctly written queries.

First, a linear multiple regression based on four parameters of the correct answer, viz., the number of columns required by a correct answer, the number of different operators, the number of rows, the number of linking variables, produced an R of 0.74 ($p < 0.001$). Addition of more Query By Example parameters would probably not increase the proportion of variance accounted for (54 percent) dramatically.

The second type of measure was the mean confidence rating of the subject on a question. This correlated with proportion correct ($r = 0.86; p < 0.001$). When combined with the above four parameters, R reached 0.92 ($p < 0.001$).

Third, two simple measures that reflected the particular English formulation of the question were calculated. One was simply the number of words in the English question. The other was the number of occasions in which an operator necessary in Query By Example was only implicit in the English phrasing of the question. For simplicity, this variable will be called "mismatches". If these predictors are included, then R reached 0.95 ($F(9,29) = 26.98; p < 0.001$).

Comparative data

Seven questions used by Gould and Ascher[12] in a behavioral evaluation of an IQF-like language were also used in the current experiment. These questions differed in the number of sets of records that were required to be retrieved (1, 2, or 3) and in the number of conjunctive modifiers required for each set (1, 3, or 5). Mean proportion correct for these questions was 0.54 for Query By Example and 0.30 for Gould and Ascher. The mean total time for these seven queries was 2.1 minutes in Query By Example and 7.7 minutes in the Gould and Ascher experiment. Subjects in the latter experiment were required to write a formulation and a plan prior to coding a query, and this coding time was 3.6 minutes. Times to write queries in the two systems were highly correlated ($r = 0.98; p < 0.001$). The correlation between the total number of modifiers and the time to write a query was significant both for the Gould and Ascher study ($r = 0.99; F(1,7) = 251.5; p < 0.001$) and for Query By Example ($r = 0.99; F(1,5) = 235.7; p < 0.001$).

Particular English wording

In order to assess the effects of the particular English wording of a question, three special sets of questions were used. Each set consisted of three different English formulations ("straight", "poorly expressed", and "rationale") that were intended to map into the same formal Query by Example. Providing subjects with a rationale did not have any apparent effect on time or accuracy. The irrelevant information in the "poorly expressed" questions produced some reduction in accuracy for all three question sets, but in no case were these differences statistically significant. Confidence ratings and times were also not consistently affected by wording of the question. In one problem, subjects took twice as long (2.2 minutes) for the "poorly expressed" question as for the "straight" version. In this case, the difference was significant ($t(76) = 4.23; p < 0.001$). It should be noted, however, that variance for the "poorly expressed" question was 15 times the variance for the "straight" version. It would seem that the inclusion of irrelevant material in a question to be translated has a large effect, but only on a subset of the population.

Interference effects

Three pairs of questions of identical objective complexity and equally well-stated were included in the experiment. For each pair, one question was included on the first 20-question test and one on the second 20-question test. The purpose of these pairs was to assess the effects of the second training sesssion (primarily how to specify universal quantification) on the ability of subjects to retain what they had learned during the first training session. There were no major differences in proportion correct, mean time, or mean confidence rating on these pairs of questions. These findings demonstrate that net fatigue, practice or interference effects were minor. However, during the second test, there were 18 cases in which universal quantification constructions were used when they were not needed.

Output classification

The output that would have resulted from each query, if an implemented system had been used, was divided into several categories (not mutually exclusive); e.g., exact output, no output, wrong output, superfluous output. The main result was that 18 percent of the 33 percent incorrect queries would produce no output. Eighty-two percent of the time that a subject made an error, some set of records would have been returned.

Error types

An incorrect query could contain more than one error. Table IV indicates the relative proportion of occasions when a particular Query By Example construction was required but was either omitted or used incorrectly. The data indicated, for example, that subjects made mistakes one-fourth of the time when they needed to use a universal quantification operator. Subjects also made mistakes on one-fourth of the occasions when they needed to use COUNT, SUM, AVERAGE, or COMPACT-COUNT operators. Overall, subjects made errors on 3 percent of the occasions in which comparison operators (e.g., >, <, ≥, ≤) were needed. Subjects made underlining mistakes on 1 percent of the occasions that they needed to decide whether to underline. The subjects made accurate decisions about "and/or" constructions and seldom used the wrong value. A common source of error not shown in the table were "mismatches" between Query By Example and English. For example, an English question might ask for the "people who have worked for us for more than five years." The tables used had an attribute labelled "Year of Hire." This question, therefore, required subjects to translate the English phrase "more than five" into "<1969".

Although not counted as an error, subjects many times left off the periods that were meant to terminate operators. Subjects were not penalized because a well-engineered implementation would include buttons that would automatically add terminators. An interesting type of error that arose on some problems was labelled "sexist syntax." For example, some subjects did not specify sex when asked to retrieve records for female secretaries.

TABLE IV—Probabilities of Various Types of Errors

Error Type	Error Probability
Quantification	0.25
Count/Sum/C-Cnt.	0.25
><≥≤	0.03
Underlining	0.01
And *vs.* Or	0.003
Wrong value	0.002

Retest data

One group of 11 subjects was asked to return after a two-week interval and six of them did. On their original tests these six subjects were slightly above average. (Their mean accuracy was 75 percent correct.) On an immediately given 20-question retest, (consisting of 10 questions from the original tests and 10 new but comparable questions) they were 53 percent correct. One subject wrote no correct queries because she forgot to underline variables. A second subject reversed the sense of '<' and '>' and wrote only 25 percent correct queries. The other four subjects were as accurate on the fourteenth day after training as they were on the first day. After an hour review of Query By Example, these six subjects wrote correctly 66 percent of their queries on a different (equally difficult) 20-question test.

DISCUSSION

Comparative data

The results indicate Query By Example is easy to learn and use in a relative sense and perhaps in an absolute sense also. Compared to the Gould and Ascher data on an IQF-like language, subjects using Query By Example required about one-third the training time, were about twice as accurate in writing queries, and were somewhat faster. Compared with Reisner's pilot data[14] on two new query languages called SEQUEL and SQUARE, our subjects required about one-third the training time and appear to be about equally accurate as those using SEQUEL or SQUARE. No subjects failed to complete training in Query By Example, unlike with these other three languages. Of course, these comparisons are only approximate because the separate experiments differed in several ways, e.g., training techniques, subject populations, test questions, stringency of scoring errors.

Absolute considerations

In an absolute sense, it is hard to imagine a powerful formal language system which could be learned much more rapidly than Query By Example. Indeed, even a "natural English" query system, which at first blush might seem to require no training time, would in fact require considerable time for subjects to learn the many exceptions and restrictions that any foreseeable "natural language" system will require. Also, the time to write a "natural language" query, and then have a computer system, through interaction with the user, disambiguate it, would probably exceed the time to write in Query By Example. There are no solid reasons for believing that accuracy in writing "natural language" queries would be substantially higher than that found here for Query By Example. Further, the fact that subjects returned in two weeks and wrote queries nearly as correctly as they did when they initially learned the language suggests that people will retain Query By Example better than they would retain a "natural language" system.

Predicting errors

The probability of a subject's making an error in writing a query is largely predictable from four parameters of Query By Example (the number of different columns used, the number of different operators, the number of rows, and the number of comparison operators) and from how confident a subject was about his query being correct. These five predictors accounted for 84 percent of the accuracy variance, but this would somewhat overstate the case in real life due to "shrinkage".[15] It should be noted that these predictions are not based upon the actual parameters of the queries that subjects wrote, but rather upon those of a correctly written query. While we have not done the former analysis, informal inspection indicates that about the same predictive accuracy would result. When the SEQUEL experiment is complete, it will be interesting to learn whether the probability of an error on a particular test question translated into SEQUEL could also be predicted from the parameters required for that question if it were translated into Query By Example.

Feedback

The practical significance of predicting an erroneous query is that a computer system can make a useful prediction on whether a user's query is in fact the one he intended. If there is a high probability of his making an error, then this prediction can trigger various kinds of feedback to the user that might help him catch his error. This is particularly important in view of the finding that a syntactic check would have only stopped 18 percent of the incorrectly written queries in this experiment. This feedback could be tailored to the type of likely error. For example, the computer might restate the user's question in another form. Or, the computer might produce alternative, but similar, queries and the user would then verify whether the one he wrote was in fact the one he intended. Or, the computer might display a small data table. Selection of particular cases within the table, either by the user or by the computer, might help in distinguishing between an intended query and an unintended one.

Weak points

Subjects made two major types of errors. First, they frequently confused SUM, COUNT, and COMPACT-COUNT operators. Presumably this error can be reduced through system constraints, feedback, and modification of the names of the operators. Second, they had difficulty in correctly specifying universal quantification when it was required. For example, subjects might render the question "List companies whose entire line of items is sold by Toys" into formal queries that really meant "List companies which have an item sold by Toys" or "List companies which supply the entire line of items sold by Toys".

The possible reasons for quantification errors are numerous, and include the possibility that subjects did not have sufficient conceptual abilities to use quantification constructions, that they were not taught that part of Query By Example well, or that they did not understand those test question statements. Alternatively, perhaps they understood the test questions and knew the language well enough, but they could not "put it all together". Perhaps they knew *how* to use the quantification constructions, but they did not know *when* to use them.

The subjects' task in translating from English into a formal language depended upon the number of "mismatches" between the particular English formulation of the question and the exact way it needed to be stated in Query By Example with the particular data tables used. This indicates that in real world settings two considerations would reduce errors. First, the data tables should be labelled to reflect the way people think about and express a subject area, and, second, the person who formulates the question in English should write the query himself or at least be aware of the ways of expressing it in the formal language. Since Query By Example can be learned so easily and queries written so quickly, it is hoped that the person formulating the question will enter his own query directly.

Good language features

In Query By Example, subjects rarely confused disjunctive and conjunctive queries, and they had little trouble linking attributes within a single table or linking multiple tables together. We observed during training that subjects sometimes seemed to understand a complex English question more easily if it were shown in Query By Example than when it was stated in English.

Positive characteristics

What are the characteristics that make Query By Example good? We are not sure, but we believe the following to be important. First, the user is given an explicit representation in which to formulate his query; he does not have to generate it free style. While this may be particularly true for the small tables (few columns) used in this experiment, it may not be true for much larger tables. (Zloof[16] has designed an extension of Query By Example to allow the user to select certain portions of a large data base which would then be queried.) Second, the *particular type* of representation given, i.e., tabular, may be especially helpful (compared with, for example, a hierarchical or set representation). Third, the system is nearly wordless. This prevents many natural language confusions, e.g., the use of an "and" in specifying disjunctive concepts ("all the physicists and children"). Fourth, the system is easy enough to learn so that people's motivations are high. (Some subjects thanked us for teaching them this language that they will probably never use in real life.) The training technique of showing subjects a series of example queries, rather than providing them with extended explanations of how the language worked, seemed quite effective. Fifth, the language is "behaviorally extendable", i.e., a novice user need only learn a small part of it to write successful queries for simple questions. Subsequently, if his problems require it, he can build upon this knowledge by learning more of the language. APL is also like this.

Query language users

Most query systems in present use have been designed for people who know their application well and regularly use the system as a main part of their job. As in the case of airline reservations systems, these users are expected to spend days or weeks learning the system because they then can use several short-cuts that lead to system efficiencies. At the other extreme is Codd's "casual user,"[17] who can be thought of as a browser or dilettante who may even sample information or problems and applications he knows nothing about. Somewhere in between are people, including "professionals", who wish to use a computer creatively and in a flexible and powerful manner on applications they know about, but without spending an inordinate amount of time learning to use the system. Our results suggest Query By Example should be especially useful for this latter group.

Research problems

Formal behavioral investigations prior to implementing a query or programming language are rare. This experiment demonstrates feasibility, and our experience is that computer scientists take seriously the results and implications. There remain many important behavioral research problems on the design and use of query systems.

In this experiment subjects *translated* a test question into a single query. In real life, people often *generate* their own questions, sometimes a related series of them, to gain the information they need. The cognitive processes involved in translating and generating questions are not identical. We plan to study how people generate questions, and we hope to learn about how people use the information they have already obtained, how they determine when they have arrived at an acceptable answer, and how the characteristics of their problem affect the types of questions they ask.

The types of questions people would ask of a data base in natural language need to be identified and compared with those that people would ask using formal languages.

Data need to be collected not only on how non-experts (as in this experiment) write queries but also on how experts would query large data bases to formulate and answer complex questions of real importance to them.

ACKNOWLEDGMENTS

We wish to thank Brian Madden for helping to grade the 1560 queries and for his many suggestions for system implementation. We also wish to thank Vivian Clingman and Cay Dietrich for their excellent secretarial services, and Bill Scholz and John Parkman for their many suggestions on this manuscript.

REFERENCES

1. Leavenworth, B. M. and J. E. Sammet, *An Overview of Nonprocedural Languages*, IBM Technical Report, RC4685, January, 1974.
2. Thompson, F. B., P. C. Lockemann, B. Dostert and R. S. Deverill, "REL: A Rapidly Extensible Language System," *Proceedings of the 24th National Conference, ACM*, Publication p. 69, September 1969.
3. Heidorn, G. E., "English as a Very High Level Language for Simulation Programming," *SIGPLAN*, 1974, *9*, 91-100.
4. *Interactive Query Facility (IQF) for IMS/360*, Publication No. GH20-1074, IBM Corporation, White Plains, New York.
5. Chamberlin, D. D. and R. F. Boyce, *SEQUEL: A Structured English Query Language*, IBM Technical Report, RJ1394, May, 1974.
6. Zloof, M., *Query By Example*, IBM Technical Report, RC-4917, July, 1974.
7. Schwartz, S. H., "Modes of Representation and Problem Solving: Well Evolved is Half Solved," *Journal of Experimental Psychology*, 1971, *91* (2), pp. 347-350.
8. Wright, P. and F. Reid, "Written Information: Some Alternatives to Prose for Expressing the Outcomes of Complex Contingencies," *Journal of Applied Psychology*, 1973, *57* (2), pp. 160-166.
9. Wickelgren, W. A. *How to Solve Problems*, San Francisco: W. H. Freeman, 1974.
10. Miller, L. A., *Programming by Non-Programmers*, IBM Technical Report, RC4280, 1973.
11. Just, M. A., "Comprehending Quantified Sentences: The Relation Between Sentence-Picture and Semantic Memory Verification," *Cognitive Psychology*, 1974, *6*, pp. 216-236.
12. Gould, J. D. and R. N. Ascher, *Querying By Non-Programmers*, paper presented at the American Psychological Association meetings, New Orleans, 1974.
13. Otis, A. S., *Otis Group Intelligence Scale*, New York: Harcourt, Brace & World, Inc.
14. Reisner, P., R. F. Boyce and D. D. Chamberlin, *Human Factors Evaluation of Two Data Base Query Languages: SQUARE and SEQUEL*. IBM Technical Report, RJ 1478, 1974.
15. Cohen, J., "Multiple Regression as a General Data-Analytic System," *Psychological Bulletin*, 1968, pp. 426-443.
16. Zloof, M., *The Invocation of Tables and Attributes*, IBM Technical Report, RC5115, November, 1974.
17. Codd, E. F., *Seven Steps to Rendezvous with the Casual User*, IBM Technical Report, RJ 1333, January, 1974.

Human factors evaluation of two data base query languages—Square and Sequel

by PHYLLIS REISNER, RAYMOND F. BOYCE and DONALD D. CHAMBERLIN
IBM Research Laboratory
San Jose, California

INTRODUCTION

Boyce et al. have recently described two data base query languages, SQUARE[1] and SEQUEL,[2] which are intended for use in an interactive mode by both programmers and professional non-programmers (e.g., accountants, lawyers, managers). The languages are comparable in the sense that the basic operators, underlying data structures and intended use are the same. They differ primarily in syntactic form, with a few additional differences in some of the specific features. Both of the languages are intended to be easily learned and used by people without specialized computer training.

This paper reports on a human factors experiment intended to evaluate the languages and, where possible in the same experiment, to compare them.

Specifically, the main goal of this experiment was to determine whether the languages could, as expected, be used by the intended populations. We were particularly interested in determining whether non-programmers, of professional caliber, could use the languages without extensive training. A second goal was to determine whether there was a difference in useability of the two languages.

A third goal, at least as important as the preceding, was to discover, prior to implementation, types of user errors that were commonly made. We felt that a language intended for non-programmers should be "user-friendly," with extensive use of the machine to catch human error. Common error types, if known, could be handled either by language changes, interactive user aids, or special emphasis in teaching.

GENERAL APPROACH

Arrangements were made with the psychology department of a local university to provide students as experimental subjects and to provide classroom space to run the experiment.

To evaluate and compare the languages, a common teaching curriculum was developed and used to teach four classes to four different subject populations: SEQUEL for programmers, SEQUEL for non-programmers, SQUARE for programmers, and SQUARE for non-programmers. Each class extended over a two week span. Extensive testing in the use of the language taught was carried out both during and after the classes.

The current experiment was preceded by a pilot experiment with a smaller subject population to debug the techniques and materials to be used in the classes.

SUBJECTS

The subject population consisted of 61 undergraduates and three graduate students from the local university, paid for their participation. They were not preselected in any known way, and thus varied considerably in background. Among the 64 participants in the experiment were students from 29 diverse majors, e.g., accounting, fine arts, recreation, mathematics, nursing, and political science.

We defined a "programmer" as anyone who had taken at least one programming course, and a "non-programmer" as anyone else. Thus there was a considerable spread of experience even within the "programmer" group. Subjects were not required to have any specific mathematics background (other than that required for admission to the university). Consequently many were unfamiliar with concepts and notation used in the languages: set comparisons, logical connectives, and even elementary mathematical notation such as "$>$."

The number of subjects completing each class was as follows: SEQUEL programmers, 18; SQUARE programmers, 11; SEQUEL non-programmers, 15; SQUARE non-programmers, 20. Several additional subjects dropped out before completion of the classes for various reasons such as illness.

THE LANGUAGES

The SQUARE and SEQUEL query languages are both based on the relational model of data proposed by E. F. Codd.[3] All information in the data base is assumed to be represented in a series of named tables, or "relations." The columns of the tables have names, and each row of each table represents information about some entity in the real world such as an employee. Figure 1 shows two example tables which describe the employees and departments of a small company.

SQUARE (Specifying Queries As Relational Expressions) utilizes a mathematical notation to express queries against a data base, while SEQUEL (Structured English Query Language) is based on English keywords. The central concept

EMP:

NAME	DEPTNO	SAL
SMITH	50	12500
JONES	50	11250
DOE	55	16000
ROBERTS	55	15500

DEPT:

DEPTNO	DEPTNAME	LOCATION
50	STAMPING	SACRAMENTO
55	MILLING	STOCKTON

Figure 1—Two relations

in each language is that of a "mapping," which returns the data-values in some column which are associated with a known data-value in another column, as illustrated by Q1.

Q1. Find the names of employees in Department 50.

SQUARE: $${}_{\text{NAME}}\text{EMP}_{\text{DEPTNO}}(\text{'50'})$$

```
SEQUEL: SELECT  NAME
        FROM    EMP
        WHERE   DEPTNO=50
```

Both SQUARE and SEQUEL allow the result of one mapping to be used as input to another mapping. This process, illustrated by Q2, is called "composition."

Q2. Find the names of those employees who work for a department located in Stockton.

SQUARE:

$${}_{\text{NAME}}\text{EMP}_{\text{DEPTNO}} \circ {}_{\text{DEPTNO}}\text{DEPT}_{\text{LOC}}(\text{'STOCKTON'})$$

SEQUEL:

```
        SELECT  NAME
        FROM    EMP
        WHERE   DEPTNO=
                SELECT  DEPTNO
                FROM    DEPT
                WHERE   LOC='STOCKTON'
```

Both SEQUEL and SQUARE allow built-in functions, including AVG, SUM, COUNT, MAX, and MIN, to be applied to any set of numeric data-values (such as the result of a mapping.)

For complex queries, the features of the two languages differ slightly. SQUARE employs a notation called a "free variable" in which the user invents a variable to represent a row, then states a test which defines the rows he wishes to be selected, as in Q3:

Q3. List the department numbers of those departments having average salary greater than 15000.

SQUARE: $$X_{\text{DEPTNO}} \in \text{EMP}: \text{AVG}({}_{\text{SAL}}\text{EMP}_{\text{DEPTNO}}(X_{\text{DEPTNO}}))>15000$$

SEQUEL avoids most uses of variables by means of a feature called GROUP BY, which organizes the rows of a table into groups by matching the values in some column, and then applies a built-in function to the rows in each group. This feature is illustrated by the SEQUEL expression for Q3.

```
SEQUEL: SELECT  DEPTNO
        FROM    EMP GROUP BY DEPTNO
        WHERE   AVG(SAL)>15000
```

Both SQUARE and SEQUEL allow use of the set-operators union, intersection and difference. Both languages permit two tables to be joined together by matching values in a common column, as in Codd's "join" operations.[3]

The teaching and testing for this experiment covered the query features of the two languages as presented in the references cited, including assignment of query results to new tables. Insertion, deletion, and update features, although they were defined for the two languages, were not included in the experiment.

TEACHING

The languages were taught in a classroom-type situation with all its attendant problems (late arrivals, Monday fatigue, classes missed for illness, etc.) The expository materials, English examples to be translated into the query language, manuals, and quizzes were the same in all four classes. The classes were highly interactive, with extensive student participation. We decided that the goal of the classes should be to teach the fundamentals of the languages to as many subjects as possible. Therefore, the pace of each class was determined by the slower members.

In general, the programmers were able to learn the languages somewhat faster than the non-programmers; hence the classes for programmers were completed after 12 academic hours of instruction, while the non-programmer classes required 14 academic hours to cover the same material.

Several example data bases were used in teaching to accustom the students to using different data bases (monopoly, school administration, computer dating, car sales).

TESTING

The major tests were: five review quizzes given at intervals during the class, a final exam at the end of the class, and a memory test given one week later. In each, a set of questions was presented in English (e.g., "Find the names of all employees who make more than their managers") and the student was required to write the appropriate query language statement. The English questions on the tests were the same for all four populations. The final exam and memory test contained forty questions each, to be completed in two hours.

The questions on the tests were carefully designed to include all features of the languages. The major emphasis of the testing was on understanding the basic features of the languages, such as mapping, composition, free variables, and set operations. Therefore, 35 of the 40 questions on the final exam were "Basic Feature" questions designed to test each of the basic features individually. The remaining five questions, called "Combination of Features" questions, tested subjects' ability to combine some of the basic features in ways they had not seen before. The memory test also contained 35 comparable Basic Feature questions, covering the same syntactic features as those on the final exam, plus five filler questions.

We attempted to write English questions for the test that were unambiguous and did not contain features, other than the ones being tested, that might create difficulties. The order of the questions on the final exam and memory test was randomized. Students were instructed that accuracy of writing queries, rather than speed, was the goal on all the exams.

The tests used data bases that had not been seen before by the students (airport administration, department store.) Each student was given a sample data base containing examples of data items in every column, for reference in formulating queries. Students were permitted to use class notes and other teaching materials on the reviews and final exam; however, the memory test was "closed book."

SCORING

Scoring the responses presented some difficulty. Obviously, a simple 'correct' or 'incorrect' decision is the simplest choice. Just as clearly, however, errors such as a misspelled word or an omitted quotation sign around a data value represent less serious misunderstanding of the language than failure to use a test on the right side of a free variable statement.

We thus devised the following scoring system:

C = completely correct
D = minor data error
M = minor language error
S = error of substance
F = error of form

Minor language errors (M) were, for example, misspelled column names or omitted quotation marks. Data errors (D) were, for example, using only a surname (Jones) when the sample data base required the full name (John Jones). These data errors were not errors that could reasonably be charged to the query language itself, since they could easily occur with any language. However, a query with such an error would not produce the correct response. We chose therefore to tabulate them separately.

Errors of substance (S) were valid forms that would run, but produce the wrong answer. Errors of form (F) were invalid forms.

In order of judged seriousness, we considered that F errors represented the most severe misunderstanding of the language, then S, M, and D in that order. Each question was given the score of its worst error. Repetitive errors were counted each time they occurred. For example, if subject 5 misspelled "personnel" in both question 10 and question 15, this was scored as two errors.

TABLE I—Mean Percentage of Essentially Correct Responses

	SQUARE	SEQUEL	Mean
Non-prog.	54.7	65.0	59.8
Prog.	77.7	77.5	77.6
Mean	66.2	71.2	68.7

a. Final Exam, Basic Functions

	SQUARE	SEQUEL	Mean
Non-prog.	52.0	60.0	56.0
Prog.	76.4	83.3	79.8
Mean	64.2	71.6	67.9

b. Final Exam, Combination of Functions

	SQUARE	SEQUEL	Mean
Non-prog.	51.4	58.1	54.8
Prog.	78.9	81.0	80.0
Mean	65.2	69.6	67.4

c. Memory Test, Basic Functions

RESULTS

Overall evaluation and comparison

To obtain a general picture of student performance, we combined the correct responses (C) with those containing minor language errors (M) and minor data errors (D) to obtain the mean percentage of "essentially correct" responses in each population. These results are shown in Table I.

The overall mean score for programmers on the final exam (counting basic feature questions only for both languages) was 77.6 percent; that for non-programmers was 59.8 percent. Analysis of variance showed that the difference in overall mean scores between programmers and non-programmers was significant ($p < .01$).

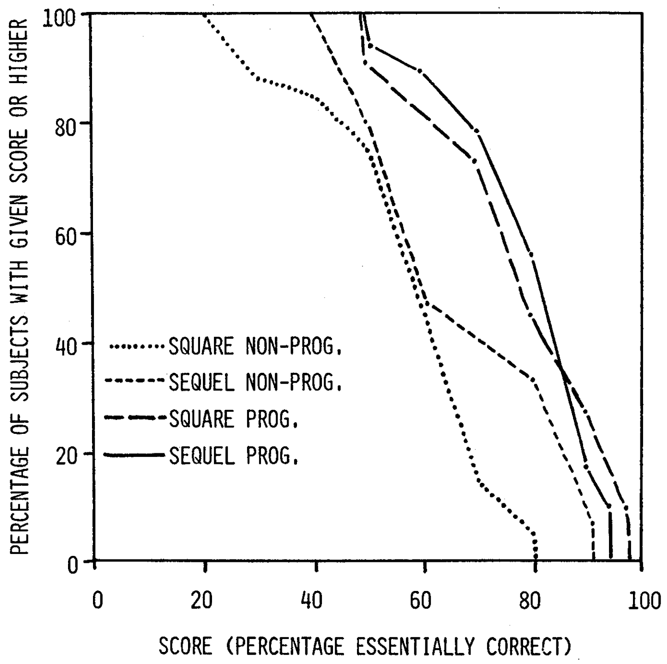


Figure 2—Cumulative frequency distribution of subject performance on final exam

Analysis revealed that neither the overall difference between SQUARE and SEQUEL nor the interaction effect was statistically significant. However, examination of Table I reveals that non-programmers showed greater facility with SEQUEL than with SQUARE, as expected. The mean scores for the final exam, basic function questions for non-programmers were 54.7 percent for SQUARE and 65.0 percent for SEQUEL; this difference was significant ($p<.05$).

Table I shows only small differences between the Basic Feature scores and the Combination of Features scores for the various populations. Subjects could apparently use the features they had learned in class to generate the novel forms required on the test about as well as they could use the individual basic features. From this we conclude that the subjects were actually expressing ideas in the new language rather than merely selecting from a set of known syntactic patterns.

In general, scores on the memory test showed no apparent decrease as compared to scores on the final exam, despite the one-week gap and despite the fact that students were not allowed to use notes on the memory test. This shows a high level of retention of the learned languages (and suggests that the final exam itself was probably a learning experience.)

Distribution of scores

Figure 2 shows the cumulative distribution of scores on the final exam (Basic Feature questions) for the four subject populations. The curves show that at least 50 percent of the test queries were expressed "essentially correctly" by the following proportions of the populations:

75% of SQUARE non-programmers
80% of SEQUEL non-programmers
91% of SQUARE programmers
94% of SEQUEL programmers

Learning curves

An attempt was made to determine the rate at which subjects learned several of the more important features of the languages, using data from the five reviews and the final exam. Figure 3 gives the learning curve for a simple mapping, showing the mean percentage of simple mapping questions expressed "essentially correctly" on the various tests by each of the four populations. It can be seen that programmers could use simple mapping in either language with over 90 percent accuracy after about two hours' instruction, and non-programmers achieved this level of proficiency after about four hours' instruction.

Learning curves for composition were also obtained but yielded more erratic patterns. On the first review testing use of composition (Review 3) the scores were:

SQUARE non-programmers:	80%
SEQUEL non-programmers:	79%
SQUARE programmers:	91%
SEQUEL programmers:	89%

Clearly, at this time, subjects understood the composition feature and could select it appropriately from the other features they had learned. After this point, scores on composition questions tended to decline, reaching the following level

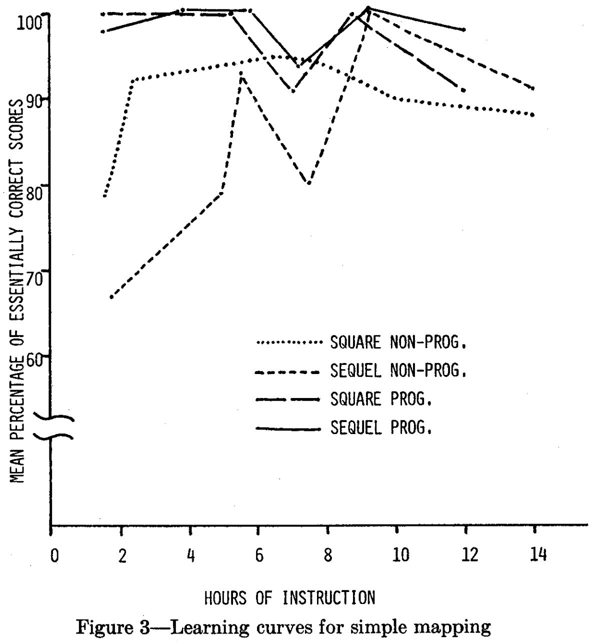


Figure 3—Learning curves for simple mapping

on the final exam:

SQUARE non-programmers:	25%
SEQUEL non-programmers:	53%
SQUARE programmers:	67%
SEQUEL programmers:	74%

Confusion generated by the later introduction of other features, such as free variables, may account for the declining ability of subjects to use composition.

Scores on questions requiring the use of free variables in SQUARE or GROUP BY in SEQUEL were much lower than those for the simpler features. Comparison of SEQUEL with SQUARE for these features should be undertaken cautiously, since the one-to-one parallelism found between SEQUEL and SQUARE for other features does not always hold between free variables and GROUP BY. On Review 5, immediately after introduction of these features, the scores for free variable and GROUP BY questions were:

SQUARE non-programmers:	30%
SEQUEL non-programmers:	33%
SQUARE programmers:	55%
SEQUEL programmers:	82%

By the final exam, the scores for these features had fallen to the following levels:

SQUARE non-programmers:	11%
SEQUEL non-programmers:	33%
SQUARE programmers:	48%
SEQUEL programmers:	48%

It appears that GROUP BY is somewhat easier to use than free variables, but both features are likely to prove difficult for non-programmers to use correctly. This conclusion is reinforced by comments made by students during the classes.

Error analysis

Table II gives the detailed breakdown of responses into five error categories for the four populations on the final exam (Basic Feature questions). It should be noted that the M and D scores do not represent the total occurrence of minor language and data errors in the experiment, because each response is placed in the category of the worst error it contains.

We observe that F-type errors, denoting inability to write syntactically correct language forms, occurred more often among non-programmers than among programmers. Also, among non-programmers, F-type errors occurred more often with SQUARE than with SEQUEL.

One of the goals of this experiment was to identify common types of user errors for which special aids could be built into the system. Study of M-type and D-type errors has yielded some preliminary results in this area.

A common tendency among subjects was to insert words from the English sentence into the SEQUEL or SQUARE query in place of the correct table name, column name, or data value. Thus, subjects wrote "NAMES" rather than "NAME," "SOLD" rather than "SALES," "PERSONNEL" rather than "EMPLOYEE," etc. These "intrusion errors" were made in both languages, by both programmers and non-programmers, in spite of the fact that subjects had sample data bases to refer to during the tests. A "user-friendly" system might be able to correct some of these minor errors by such techniques as word-stem matching or alternative-form dictionaries.

TABLE II—Percentage of Responses in Each Response Category on Final Exam, Basic Feature Questions

Response Category	SQUARE Non-Prog.	SEQUEL Non-Prog.	SQUARE Prog.	SEQUEL Prog.
C (Correct)	42.7	56.2	66.0	68.2
D (Minor Data Error)	2.2	1.9	2.1	1.3
M (Minor Lang. Error)	9.9	6.9	9.6	7.9
Total Essentially Correct	54.8	65.0	77.7	77.4
S (Error of Substance)	11.5	10.9	10.6	10.2
F (Error of Form)	33.8	24.2	11.7	12.4
Total Incorrect	45.3	35.1	22.3	22.6

Misspellings of keywords, table names, and column-names were also common, despite the availability of the sample data base. This suggests that a spelling corrector would be another important part of a "user-friendly" system.

DISCUSSION OF METHOD

An early decision was made that the SQUARE and SEQUEL classes for this experiment would be taught in a conventional classroom situation with a human teacher rather than by a system of programmed or computer-assisted instruction. This decision was made because classroom teaching is relatively quick to implement and known to be effective, and because it provides opportunity for on-the-spot feedback between the teacher and the student. On the other hand, a programmed instruction or CAI curriculum would have been more reproducible. Furthermore, an individualized curriculum allowing each student to learn at his own rate and to take an examination when ready would have provided a more accurate measure of learning times for individual students.

All testing for the experiment was done on paper in the classroom. If an actual interactive data base management system had been available for testing, it might have been able to provide feedback to prevent subjects from making the same error repeatedly. On the other hand, such a test would confuse difficulty in use of the language with difficulty in learning to use the terminal system.

A limitation of the experiment was that the effectiveness of the teaching techniques and the learnability of the languages tend to be confounded. Since the results of the experiment were favorable, we conclude both that the languages are reasonably easy to learn and that the curriculum was reasonably effective; however, it is difficult to separate the two factors. To some extent, the goals of the experiment were conflicting (e.g., we wished to study user errors, but if the teaching were perfectly effective, errors would be eliminated).

NEED FOR FURTHER WORK

The field of language evaluation is in its infancy and provides fertile opportunities for further research. The present experiment focused on the learnability of languages, but other language aspects remain to be measured; for example, the ease with which one subject can understand a query written by someone else, or the ease with which a query can be modified.

In our experiment, subjects were unsystematically assigned to classes with the expectation that individual differences among subjects would "balance out" and allow the different classes to be compared (e.g., SQUARE class vs. SEQUEL class for non-programmers). This aspect of the experiment could be made more rigorous if an "aptitude measure" were developed which would enable matching the aptitudes of students in two populations before the experiment is begun.

SUMMARY

A series of experiments was conducted to evaluate the learnability of two data base query languages, SQUARE and SEQUEL, using university students as subjects. The students were divided into two groups according to whether they had had programming experience. Experiments showed that both populations were able to use either language with reasonable proficiency after 12 to 14 academic hours of instruction (lowest class mean was 55 percent for SQUARE non-programmers; mean score for programmers was 77 percent in both languages). Programmers learned both languages more quickly and more completely than did non-programmers, and the non-programmers showed greater proficiency with SEQUEL than with SQUARE. Test scores showed that both programmers and non-programmers were able to combine the basic language features in ways they had not been explicitly taught. Also, it was shown that, one week after the end of instruction, subjects still retained nearly all their proficiency and were able to use the languages successfully on a closed-book examination.

The individual features of the languages varied considerably in learnability. The basic language feature, a simple mapping, was learned in each language with near-perfect accuracy by programmers after two hours and by non-programmers after four hours. However, considerable difficulty was experienced in learning and retaining the more complex "free variable" and "GROUP BY" features, especially by non-programmers.

A study of errors made by subjects suggests that a real data base system should be prepared to correct minor syntactic errors and to search for poorly-specified data values by some technique such as stem-matching or a synonym dictionary.

REFERENCES

1. Boyce, R. F., D. D. Chamberlin, W. F. King, and M. M. Hammer, *Specifying Queries as Relational Expressions: SQUARE*, Research Report RJ 1291, IBM Research Laboratory, San Jose, California, October 1973. (To appear in *Communications of the ACM*.)
2. Chamberlin, D. D. and R. F. Boyce, "SEQUEL: A Structured English Query Language," *Proceedings of ACM SIGFIDET Workshop*, Ann Arbor, Michigan, May 1974.
3. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, *15*, June 1970 pp. 377-387.

BIBLIOGRAPHY

1. Gould, J. D., *Query by Non-programmers*, paper presented at the 82nd Annual Convention of the American Psychological Assoc., New Orleans, August 1974.
2. Thomas, J. C. and J. D. Gould, *A Psychological Study of Query By Example*, Research Report RC 5124, IBM T. J. Watson Research Center, Yorktown Heights, N. Y., November 1974.
3. Weinberg, G. M., *The Psychology of Computer Programming*, New York, Van Nostrand Reinhold Company, 1971.
4. Young, E. A., "Human Errors in Programming," *International Journal of Man-Machine Studies*, 1974, *6*, pp. 361-376.

A classification of compression methods and their usefulness for a large data processing center

by DORON GOTTLIEB, STEVEN A. HAGERTH, PHILIPPE G. H. LEHOT and HENRY S. RABINOWITZ

Fireman's Fund American Insurance Companies
San Francisco, California

INTRODUCTION

The compression techniques surveyed in this paper all work to reduce storage space for data files at the price of increased CPU activity needed for compression and decompression. As CPU time becomes cheaper relative to the cost of external storage devices, compression appears as an increasingly attractive option for dealing with large files.

In a small shop, which is typically I-O bound, compression uses available CPU time to decrease the amount of disc or tape storage. More generally, compression of storage space is achieved only at the expense of CPU time. The most clear-cut use of compression is for archive files where the main consideration is minimizing physical storage space.

This paper surveys available techniques for automatic reversible compression of files; i.e., techniques that require no special knowledge of the contents of a file. The theoretical advantages of the two main categories of compression—differencing and statistical encoding—are compared, and the practical results of these techniques on large insurance files are shown, both in terms of compression efficiency and CPU efficiency.

Suggestions are offered for improving the compression achieved through Huffman coding by adding a schema to code strings of a repeated character. An Algorithm is given to find the threshold for the minimal length of those strings whose coding will result in improved compression.

DEFINITIONS AND USES OF COMPACTION AND COMPRESSION

Definitions

Since there are no standardized definitions of compaction and compression, we propose the following usage, to be followed throughout this paper:

Compaction of data means any technique which reduces the size of the physical representation of the data while preserving a subset of the information deemed "relevant information."

Compression of data is a Compaction technique which is completely reversible.

Compression ratio is the size of the compressed file expressed as percentage of the original file.

A compaction technique that is not a compression technique involves elimination of information deemed superfluous in order to decrease overall storage requirements. Such a technique is, by definition, dependent on the semantics of the data.

The file-oriented techniques studied in this paper are primarily compression techniques since these are the easiest to implement in a generalized fashion. While a familiarity with the semantics of a file is necessary for maximal compaction, compression techniques have the advantage of "automatic" applicability to a wide variety of files.

Compaction techniques that are irreversible are most applicable to directories of a file. Indeed, at a directory level there is often an advantage in disregarding less important information, which may be carried in lower directories or in the file itself, so as to speed up the directory scanning and the overall efficiency of the "general directory access method."

COMPACTION OF A SEQUENCE OF SORTED RANDOM KEYS

Introduction

A good example of compaction is the following front-compression/rear-compaction scheme on a sequence of sorted keys. The scheme achieves a very compact first level directory in which only those portions of a key K are kept that are

—not identical to the previous key
—necessary to make K unique; i.e., distinct from previous key and following key.

In particular, the "front string" (the initial string of characters of K identical to the same-positioned characters in the key before K) will be skipped. The "rear string" (the string of trailing characters which are not needed to distinguish K from the previous key and the following key) is knocked out. Rear compaction involves a loss of information. Hence, the keys must be carried with their full information at the level of the record, or at some intermediary level.

Front compression

The leading bits of a key which are identical to the previous key's leading bits constitute the FRS (front redundant string) and need not be repeated. The FRS is expanded to include one extra bit, since it follows automatically that if the first n bits of a key are the initial repeated string, then the $(n+1)$st bit must be different. Instead of the FRS itself, a number can be written specifying the length of the FRS. This number will only require a field of bits equal to the logarithm (base 2) of the length of the key. For example, if m is the length of the key, say $m=32$ bits, then the length of FRS cannot exceed m; hence, the number of bits needed to express the FRS-length is $[\log_2 m]=5$ bits.

Rear compaction

Unlike front compression, which suppresses some redundancy but does not really do away with any information *per se* (provided one knows the previous key) the rear compaction will do away with information which is judged unnecessary.

Rear compaction will delete an "RRS" (Rear Redundant String). An RRS is composed of those right most bits of a key which are not necessary to uniquely distinguish this key with respect to the set of all keys in the particular sequence to be rear-compacted. We can immediately state the following theorem:

Theorem:

Given a set of keys, some of which may be identical, in order to find the RRS of a key K, it is enough to look at the previous key P and the following key A in the sorted sequence; i.e., the RRS for K relative to the whole set of keys is identical to the RRS for K relative to the set of 3 keys P, K, and A.

The useful string (US) is what is left of the key after the FRS and the RRS have been removed.

Example:

P Key # $(i-1)$ 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

FRS US RRS

K Key # i 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0

A Key # $(i+1)$ 1 0 1 0 1 0 1 1 0 1 0 1 1 1 0 1 1

Key # i will be coded as (8) 0 1 0 1 0, where 8 is the size of the front redundant string (FRS) and "01010" is the useful string left over after front and rear compaction.

Note that the last bit, and only the last bit, of the FRS differs from the corresponding bit of the previous record. Note also that the FRS of key # i is sufficient to distinguish it from key # $(i-1)$, but that the 5 bits of the Useful String (US) are needed to distinguish key # i from key # $(i+1)$.

Note: Another way of looking at rear compaction is to define the useful string of key # i as follows:

> If FRS of key # $(i+1)$ contains the FRS of key # i then the US of key # i is the string obtained by deleting the FRS of key # i from the FRS of key # $(i+1)$; otherwise it is null.

If the keys are viewed as binary numbers of n bits each, then the compacted key # i will occupy $[\log_2(K_{i+1}-K_i)]+[\log_2(K_i-K_{i-1})]+2[\log_2 n]$ bits in the first case and $2[\log n]$ bits in the second.

COMPRESSION BY DIFFERENCING

The term differencing describes techniques which compare a current record to a pattern record and retain only the differences between them; i.e., information in compressed record = information in current record—information already in pattern record.

This technique is particularly successful with large record files of alphanumeric characters where most corresponding fields in different records are the same (or even blanks or zeros); also, compression is often improved by sorting the file on the largest field.

In some sense, the differencing scheme is a generalization of front compression seen above. The process of compressing the front string is repeated for each maximal substring in the current record which matches a substring (in the same position) in a pattern record. The start and end signals for such a matched substring are the overhead for the scheme. The information unit on which differencing is performed can be the bit, the byte, the field, or logical information.

a. Bit: Both current and pattern records are considered as equal length bit strings. (They could also be left or right justified variable-length bit strings.)
b. Byte or character: Both current and pattern records are viewed as character strings. (Byte access being cheaper, this is the most common case.)
c. Field: The record is viewed as a string of fields (each with its own characteristics). Quite often, the start and end signals for the unmatched "strings" will be implemented by bit maps, where each bit for the map is on or off to signal whether a given field of the current record is identical to the corresponding field of the pattern record. It is a rougher scheme, but it may

present the advantage of less over-head whenever matching fields are frequent.

d. Logical information (instead of physical data such as bit/byte/field) as in the example:

date 1, date 2, date 3 (=) date 1, interval 2, interval 3
where interval 2 = date 2—date 1
interval 3 = date 3 − date 2

In conclusion, we see that differencing schemes, (like front compression which is a special case) seek to diminish the overall amount of information by not repeating (and actually subtracting) that part of the information in a record which is already present in another (previous/pattern) record.

Most often, differencing is applied to sequential files where the pattern record is taken to be the previous record in the file, which itself may have been sorted.

If used with a direct access file, the first record of the block which directly accessed should be left intact (non-compressed). This may be expensive when the ratio (size of non-compressed record)/(size of block) is not small enough. In this case, a change in the blocking format might be warranted.

Zero and blank compression techniques can be viewed as a special case of differencing in which a zero or blank record is used as the pattern record for the entire file.

The use of the same pattern record for the whole file may not yield as good a compression as a schema where the pattern used to compress each record is the record preceding it. But the latter choice is more expensive in encoding and decoding time. Indeed, whenever a record is to be read every record preceding it has to be decoded; i.e., half a block decompression on the average. Deletions and insertions are, clearly, even costlier.

The Ling-Palermo algorithm for compression of blocks of data[1] through a clever use of linear dependence concept is an extension of differencing and so is the QUATREE method by Hardgrave.[2]

STATISTICAL ENCODING

A statistical encoding is a transformation of the user's alphabet, converting each member of the alphabet into a code bit string whose length is inversely related to the frequency of the member in a text.

A text is normally written using a fixed alphabet where each character is represented by a fixed length bit string (e.g., a byte). A statistical encoding schema attempts to take advantage of the fact that different characters will usually occur with different frequencies. Coding each character as a bit string of length inversely related to its frequency (i.e., coding non-frequent characters with long ones) will usually compress the text.

If a text is written in an alphabet $I = \{\alpha_1, \ldots, \alpha_n\}$ where each character occupies k bits, then an efficient statistical encoding will assign a code β_i to each character α_i such that:

$$\sum_{i=1}^{n} | \beta_i | \leq k^*N$$

where f_i is the frequency of α_i in the text, $| \beta_i |$ is the length of the code β_i, and N is the number of characters in the text.

An essential property of any statistical encoding schema is complete reversibility. That is, the ability to retrieve the original text from the encoded one in a finite (preferably linear) number of steps. Another desired quality is the prefix property where no code β_i is the prefix of another code β_j. This property assures both complete and unique reversibility and also that the decoder never has to back up and rescan any portion of the text. It is sometimes desired to have a coding schema that will preserve the alphabetic ordering of the user's alphabet (the alphabetic property) that is, if α_i precedes α_j in the original alphabet, one should be able to deduce this fact from the codes β_i and β_j without having to decode them.

Information-Theoretic considerations assure us that when an alphabet of n characters is coded so that its complete reversibility is assured, $N*H$ is the shortest possible binary representation for a text of N characters where H is the entropy of the distribution of characters in the text. This means, roughly, that the more "skewed" the distribution, the better the compression.

Huffman coding scheme[3] is a very elegant and simple statistical coding algorithm with the prefix property. It is optimal in the sense that its performance reaches the information-theoretic lower bound stated above. The Hu-Tucker algorithm[4] is a statistical coding scheme with both the prefix and alphabetic property. In the next section, we discuss in more detail the application of these two algorithms.

EVALUATION OF HUFFMAN CODING FOR LARGE BUSINESS FILES

Huffman coding, based on statistical characteristics of a file, provides an easy and effective method of file compression without necessitating any inquiry into the semantics of file records. Thus, one package can be used on a wide variety of files to achieve compression without investment of large amounts of programmers' time to investigate particular files for their storage-wasteful properties. In testing a Huffman encoding package on a variety of large insurance files, the worst results encountered (on an already compact binary file) were 50 percent compression.

Furthermore, contrary to Kreutzer,[5] Huffman coding is suitable for files that are frequently updated. This is because the compression ratio achieved by Huffman coding can be discerned immediately from a table of the frequency of occurrence of each character in the file. If the frequency of occurrence of letter i is f_i, then the expected code length generated by Huffman code is closely approximated by the entropy of the frequency table $H = \sum_{i\in I} [(f_i/\sum_{i\in I} f_i)^* \log_2((\sum_{i\in I} f_i)/f_i)]$ where I is the alphabet being coded. In fact, $H \leq$ expected code length $\leq H+1$. Thus, if a file is frequently updated, it is easy to compile new frequency statistics at the same time updates are performed. Then, after a certain number of updates, the expected code length using the new statistics and a new coding can be compared with the expected code-length using the new statistics and old code. If a significant improvement can be made by recoding (which will probably occur only rarely), then a new code can be generated and the file recoded.

Because the compression-ratio can be calculated using a simple statistical pass of the file, the Huffman technique gives more immediate information than a differencing technique which cannot give an advance notice of its effectiveness before an actual encoding pass occurs. Huffman coding has the further advantage, over differencing techniques, that records can be decoded individually without need for some reference to a pattern record. In fact, differencing techniques derive most of their power from the fact that blanks or zeroes are commonly repeated, a fact that is handled well by Huffman coding, and even better by a modification of Huffman coding that is discussed below.

It is hard to improve on Huffman coding while still preserving its "automatic" effectiveness; i.e., without reference to data semantics. However, some progress can be made in special coding techniques for repeating characters and unrecognized characters.

Huffman code is optimal given the assumption that the probability of appearance of any letter is independent of the probability of appearance of any other letter. Of course, this is never actually the case, but this assumption is necessitated by the difficulty of discerning patterns in an automatic fashion. The simplest "pattern" is a repeating string of the same character (a clump). Usually the most frequent character in a file (say, blank or zero) is not randomly distributed throughout the file but occurs in clumps. Since this commonly occurring condition violates the assumptions under which Huffman is optimal, it is possible to devise strategies to improve on Huffman, the simplest of which is to invent a "repeat" flag.

A repeat flag can be included in the frequency table of a file with frequency equal to the number of occurrences of repeat strings whose length is greater than some threshold T. Then, for example, the string b̸b̸b̸b̸b̸, instead of being encoded as (code b̸) (code b̸) (code b̸) (code b̸) (code b̸), will be encoded as (code of repeat flag) (5) (code b̸). (Note the introduction of the repeat flag modifies the frequencies of the characters which are repeated beyond the threshold.) Despite the vicious cycle nature of the problem, there is an algorithm that enables one to estimate the lower threshold T for length of repeated strings above which use of the repeat flag is more efficient than simple Huffman code. This algorithm depends only on the frequency of occurrence of characters and of repetitions. In practice, it turns out that this technique provides significant improvement over Huffman only when applied to the most frequent character in the file.

We will assume here that repeating strings of character α_i are to be encoded using the format:

(code of repeat flag) (clump size)

The following tables are obtained by scanning the file once:

Character	Frequency
α_1	f_1
α_2	f_2
⋮	⋮
⋮	⋮
α_n	f_n

$$N = \sum_{i=1}^{n} f_i$$

TABLE A
Frequency of Characters

Clump Length	Frequency of Clumps
m	φ_m
$m-1$	φ_{m-1}
⋮	⋮
2	φ_2

TABLE B
Frequency of Clumps of Character α_i

Where f_j is the total frequency of character α_j in the text and φ_k is the number of clumps (of character α_i) of length exactly k (i.e., k successive occurrences of α_i bounded on both sides by different characters). The threshold T is then the maximal K satisfying:

Length of flag+count field $\leq K$*length of code for α_i

Denoting by α_{n+1} the flag character, we will use $\log_2(N/f_j)$ as an estimate for the length of the Huffman code for character α_j, and $[\log_2 m]$ as the number of bits in the fixed size count field (m is the size of the longest clump of α_i in the text).

The following is the algorithm to find T:

(1) set $r = o$

(2) set $r = r+1$

(3) set $f_i = f_i - (m-r+1)*^{\varphi}m-r+1$ (adjusting the frequency of α_i by subtracting occurrences of α_i in clumps of size $m-r+1$)

(4) set $f_{n+1} = f_{n+1} + \varphi_{m-r+1}$ (adjust the frequency of the flag)

(5) set $N = \sum_{j=1}^{n+1} f_j$ (adjust the total)

(6) Evaluate:

$$\log(N/f_{n+1}) + [\log_2 m] < (m-r+1)*\log_2(N/f_i)$$

If the inequality holds, go to Step 2. Otherwise, readjust

$$f_i = f_i - (m-r+1)*\varphi_{m-r+1},$$

$$f_{n+1} = f_{n+1} - \varphi_{m-r+1},\ N = \sum_{j=1}^{n+1} f_j$$

(i.e., use the frequencies from the previous step).

Set $T = m-r+2$ and proceed to produce Huffman code using the resulting Table A. Encode the file applying the repeat-flag format to α_i clumps of size $\geq T$.

The above algorithm could be modified so that clumps of other frequent characters could be evaluated. This will require a table (like Table B) for each of the characters under consideration and possibly a format:

(flag) (count) (code of repeating character)

Note that to obtain the expected gain in compression one could, at Step 6, compute the entropy of Table A and use that to compute the compression ratio.

Another simple addition to Huffman coding is the unrecognized character flag. Suppose a file is to be encoded byte by byte, as is natural with IBM implementations. In most files, many of the 256 possible patterns of 8 bits do not occur. If these patterns are included in the coding, even with a weight of zero, space needed to store the code table will increase. Thus, it is more efficient in terms of storage (and CPU time for code making) to code only those characters that actually appear in the file, along with a special flag to mark the presence of an unrecognized character. Then, if a character not in the code table becomes included in the file due to an update, it will be coded by the code for the unrecognized-character-flag followed by the character itself written as 8 bits.

Unfortunately, this technique is not suitable to Hu-Tucker coding. Hu-Tucker coding is nearly as short as Huffman coding and it preserves alphabetical order. Thus, it would seem to be useful in a situation where alphabetical sorting of records or keys would be necessary. However, if the file is to be updated at all, the user is faced with two equally unpleasant alternatives. One is to code every possible character that could ever occur. Even if the absent characters were coded with frequency zero, this would greatly increase the expected code length. (Unlike in the Huffman code tree, the unused characters cannot be stuck off in one remote subtree, but must be interspersed with the other characters in natural alphabetical order, thus, increasing the code length for all.) The other alternative is to use an unrecognized character flag. But this technique destroys the alphabetic property which distinguishes Hu-Tucker. Thus, Hu-Tucker coding is of practical interest in files that are rarely, if ever, updated or whose character set is fixed.

In conclusion, Huffman coding is the optimal prefix property bit string coding given a particular choice of alphabet. However, due to patterns and dependencies among the data, the choice of alphabet itself can make a difference in the efficiency of the coding.

CONCLUSIONS

A variety of compression techniques were applied to large insurance files some of which were already in compact form (that is, after a semantic analysis was used to eliminate "redundant" information like long strings of blanks, etc.). The programs were all written in PL/1 and executed on IBM 370/168 system. CPU measurements given below have only a relative meaning. For production purposes, assembler-code routines should perform roughly 10 times faster.

Differencing

Differencing techniques are limited to files of fixed formatted records. Differencing was the most economical method as far as CPU time. Encoding required about 5 milliseconds (mls) per 1000 characters, and decoding about 3 mls. Sequential differencing, where each record is used as a pattern for the record succeeding it, yielded good compression ratios varying between 28 and 44 percent. However, the disadvantages of this technique are apparent. Any update requires a complete decoding of the entire file since the code for every record depends on *all* records preceding it. This also implies that physical damage to a record will propagate and might hinder complete decoding of succeeding records. Trying to get around this by using a fixed pattern for the entire file (or fixed pattern for each block) alleviated the problem at the expense of yielding worse compression ratios that ranged around 45 percent.

Differencing is also characterized by the fact that there is no need for a scanning pass of the data before actual encoding. This, however, implies that one cannot automatically predict the compression ratio without actually encoding the file.

Huffman coding

Huffman coding, unlike differencing, can be applied to variable length as well as fixed length records. Huffman coding achieved good compression ratios (between 35 and 49 percent) and was surpassed only on one file by sequential differencing. The CPU time for scanning the file and obtaining the frequency table was negligible. The production of the code table from the frequency table required less than 50 mls. We have observed that it was enough to sample only 3-5 percent of a file in order to obtain frequency tables and produce a code table which was identical to the one obtained by scanning the entire file. The cost in CPU time of encoding and decoding was at about 100 mls per 1000 characters, or roughly 20 times more than differencing (in the case when differencing is applicable). This fact is attributable to bit level versus byte level manipulation.

Applying Huffman coding together with the repeat-flag schema improved the compression ratio to between 28 and 43 percent without any detectable change in CPU cost for encoding or decoding.

Huffman coding requires an initial statistical pass through the file or through part of it. The frequency table obtained from the initial pass gives an excellent indication of the compression ratio that could be achieved if the file is to be compressed using Huffman coding; i.e., the user could decide whether it is worthwhile compressing a file without the need for an actual compression run.

The frequency table that is attached to the file can be updated continuously with every deletion and insertion to the file (at negligible CPU cost) so that the actual compression ratio of the file and the maximum achievable compression are always available to the user or to an automatic monitoring routine for deciding whether a new code table should be produced.

Hu-Tucker

The Hu-Tucker code, as was mentioned in the previous section, is a statistical code which preserves alphabetic ordering.

The CPU costs of using Hu-Tucker coding are the same as those for Huffman coding. The compression ratios achieved were only very slightly worse than Huffman. The decline in compression (as compared to Huffman) never exceeded 7 percent.

Hu-Tucker coding is especially useful for directories and files where frequent sorting is necessary. The alphabetic property enables the user to sort a compressed file without the need to decompress it.

In short, we found statistical compression methods to be more generally applicable than differencing to a variety of file structures, alas, at a cost of higher CPU time.

REFERENCES

1. Ling, H., and F. P. Palermo, *A Block Oriented Information Compression,* IBM San Jose Research Center, Report RJ 1172, No. 19024.
2. Hardgrave, W. T., *The Prospects for Large Capacity Set Support Systems Imbedded within Generalized Data Management Systems,* International Computing Symposium, Davos, Switzerland, Sept. 4-7, 1973.
3. Huffman, D. A., "A Method for Construction of Minimal Redundancy Codes," *Proc., I.R.I.E.,* 51, pp. 1098-1101, Sept. 1952.
4. Hu, T. C., and A. C. Tucker, "Optimal Computer Search Trees and Variable Length Alphabetical Codes," *S.I.A.M. Journal of Applied Mathmetics,* 21, 514 (1971).
5. Kreutzer, P. J., *Data Compression for Business Applications,* Navy Fleet Material Support Office.
6. Tunstall, Brian, "Synthesis of Digital Compression Codes," *Hawaii International Conference on System Sciences,* Jan. 1968, pp. 266-268.
7. Tunstall, Brian, *Synthesis of Noiseless Compression Codes,* Research Report #67-7, Georgia Institute of Technology.
8. DeMaine, P. A. D., *Principles of the NAPAK Alphanumeric Compressor in the SOLID System,* National Bureau of Standards, Tech. Note 413, August 15, 1967, Part III.
9. Gilbert, E. N., and E. F. Moore, *Variable Length Binary Encoding,* The Bell System Technical Journal, July 1959.
10. Ott, Eugene, "Compact Encoding of Stationary Markov Sources," *I.E.E.E. Transactions on Information Theory,* Vol. IT-1, No. 1, Jan. 1967.
11. Rottwitt, Theodore, Jr., and P. A. D. DeMaine, *"Storage Optimization of Tree Structured Files Representing Descriptor Sets.*
12. Rice, R. F., *The Code Word Wiggle: TV Data Compression,* Technical Memorandum 33-428, National Aeronautics and Space Administration Jet Propulsion Lab., Cal. Tech., Pasadena, Calif., June 1969.
13. Knuth, D. E., *The Art of Computer Programming,* Vol. 3, Addison-Wesley.

A simulation model for data base system performance evaluation

by FUMIO NAKAMURA, IKUZO YOSHIDA and HIDEFUMI KONDO
Hitachi Limited
Yokohama, Japan

INTRODUCTION

Performance evaluation represents one of the most critical and also most complex aspects of the design of a data base system for operation in an on-line environment.

Most techniques for computer system evaluation (such as software and hardware monitoring) presume that the system or at least a skeletal version of it is operational. Such techniques are useful primarily for turning already developed systems.

Simulation, on the other hand, although it is expensive, offers a way to evaluate a system with relative accuracy prior to its development. By varying design parameters, the system designer can hope to identify potential bottlenecks, avoid costly design mistakes, and eliminate some of the guess work of identifying the most suitable system solution.

The subject of this paper is a large scale simulation model which was developed for data base system performance evaluation. The model, which is used for evaluation of design alternatives for application systems as well as system software (OS(Operating System) and DBMS (Data Base Management System)), employs event-driven simulation of the actual operations of hardware, host operating system, DBMS, and application programs.

Great care was taken to model OS and DBMS functions, such as task scheduling, I/O interrupt processing, communication processing, message scheduling, data base access processing, buffer management, and disk space management independent of any application characteristics.

In order to evaluate an application data base system, description of hardware configuration and data base characteristics, as well as application program models, must be prepared. Modifications of the actual model are required only if alternative OS and/or DBMS architectures are being examined.

Once the user has developed a model of his system, its behavior can be evaluated with relative ease, by simply modifying parameters such as message traffic, data base buffer set size, number of concurrent application program tasks, data base organization, and disk storage allocation.

The remainder of this paper gives a presentation of the model, discusses some simulation results, and outlines approaches for further development.

SIMULATION MODEL

The simulation model written by using a computer system simulation package has two major portions, definition part and procedure part. The definition part describes the system environment being simulated, while the procedure part represents the software system using an instruction set prepared by the simulator. The simulator executes instructions in the procedure part interpretively referencing information in the definition part.

Our model consists of the following components.

Definition Part
(1) Hardware description
(2) Application description
Procedure Part
(3) Operating system model
(4) Data base management system (DBMS) model
(5) Application program models

For an application system, (1), (2) and (5) must be prepared to be input into the simulator. Figure 1 shows an overall view of the model.

The following is obtained through simulator runs.

- Resource utilization (CPU, channels, disks, magnetic tapes, lines, etc.)
- Frequency of each software module usage
- Response time and throughput
- Number of data base I/Os
- Queue statistics

From the above information, the designer can estimate the capacity of his system, identify bottlenecks and critical factors, which enable him to decide improvement steps, and consider the trade-offs.

Hardware description

It includes configuration description and definition of the following hardware characteristics.

- CPU—instruction mix (CPU speed)
- Lines—transmission rate and delay

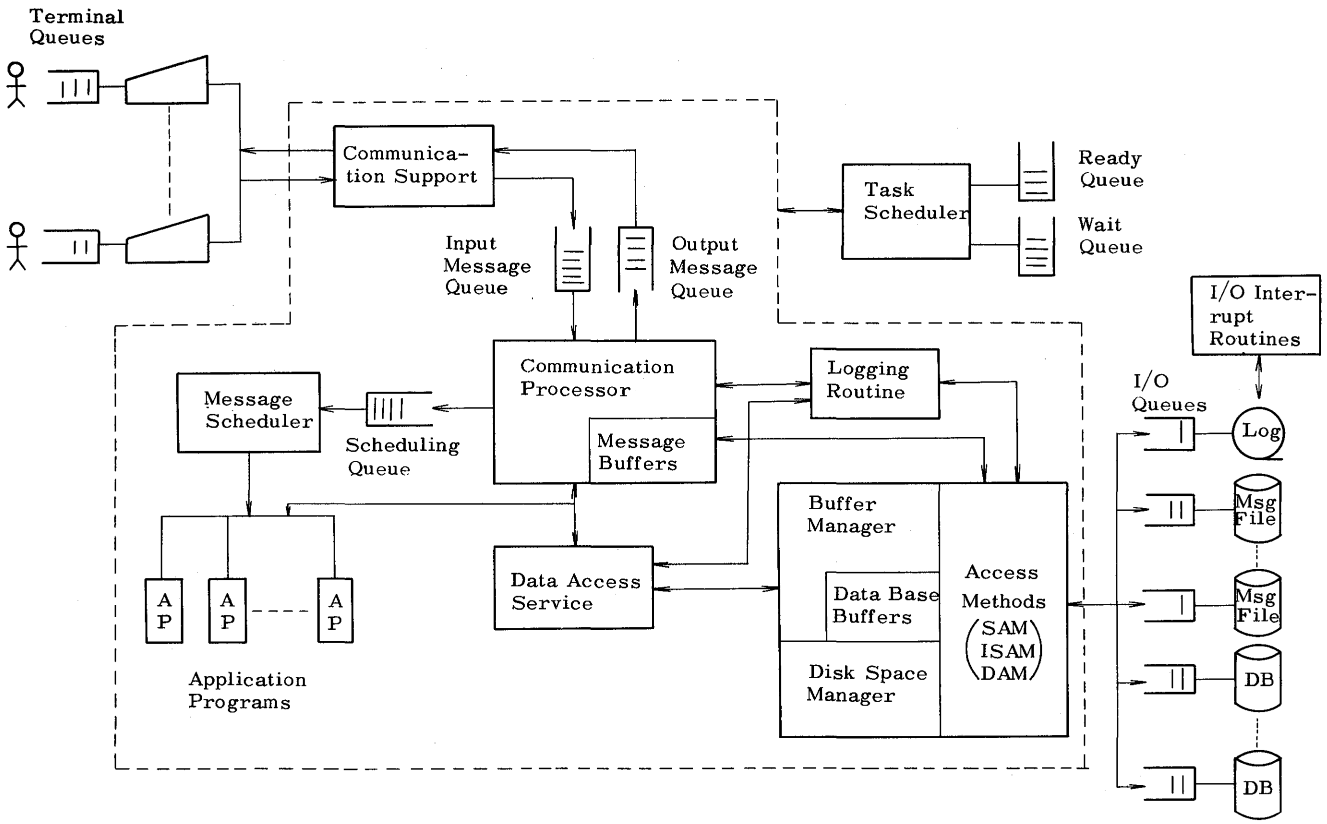


Figure 1—Overall view of simulation model

- Terminals—message generation rate and distribution
- Channels—data transfer rate and channel interference rate
- Disks—data transfer rate, cycle time, seek time characteristics and RPS (Rotational Position Sensing) lead and hold time
- Magnetic tapes—data transfer rate and start time

Application description

The most important information here is the data base definition characterized by data structures, storage structures and access methods. Figure 2, Table I and Table II show how we are dealing with data base definition in the model.

(1) Data structures—The DBMS supports hierarchical data structures, composed of an aggregate of elements called segments. Each segment has one parent segment except a root segment and any number of child segments. Item 6 to the last in Table I describe a data structure.

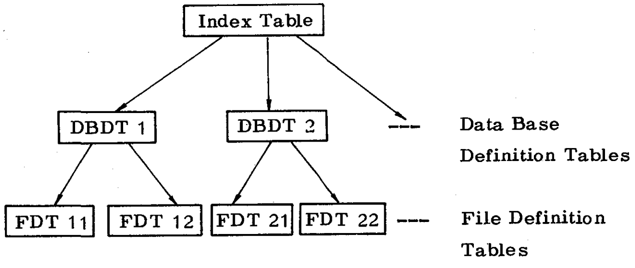


Figure 2—Table relationships in data base definition

(2) Storage structures—Item 2 and 5 in Table I and item 2 to 7 in Table II are concerned with a storage structure which, due to its nature, cannot be easily represented in a model; that is, there may be tens of thousands of records or millions of records and a model can never keep the exact position of each record except environmental information such as disk type, block length, and file allocation on secondary storage. The only information related to the above problem is the segment overflow ratio in Table I and this information and some additional data base access calls will make up for the problem. (See DBMS model)

(3) Access methods—The DBMS supports, as basic access methods, Sequential Access Method (SAM), Indexed Sequential Access Method (ISAM) and Direct Access Method (DAM).

Operating system model

Excluding comments, 400 simulator instructions were required to model the following operating system functions.

(1) Task scheduler—A data base system in an on-line environment is executed in multi-programming and/or

TABLE I—Data Base Definition Table

No.	CONTENTS
1	Data Base Name
2	Disk Type to Store Data Base
3	Pointer for FDT 1
4	Pointer for FDT 2
5	Segment Overflow Ratio
6	Number of Segments in Data Base
7	Segment Name
8	Segment Length
9	Segment Level in Hierarchy
10	Parent Segment Name
11	Occurrence under Parent Segment
⋮	Repeat 7 ~ 11 until Last Segment

multi-tasking modes, which requires a module for task management. This module attaches and detaches tasks to the DBMS and application programs, maintains ready and wait queues, and activates and inactivates tasks according to their priorities.

(2) Communication support—This component manages communication lines and terminals, controls polling, sends and receives messages and services message I/O completion interrupts. A productive poll which follows message transmission occurs when a terminal generates a message. After message transmission has completed, the message is put into the input message queue. An output message in the output message queue, on the other hand, is sent to a terminal, which causes an output transmission completion interrupt that restarts polling.

(3) I/O interrupt service routines—The model has I/O interrupt routines according to I/O device types which involve magnetic tapes (for log data), HITAC 8578 Disk Storage (cycle time 25 ms, average seek time 60 ms, capacity 29 MB) and HITAC 8589 Disk Storage (cycle time 16.7 ms, average seek time 30 ms, capacity 100 MB). Completion of read, write and seek gives control to these routines.

DBMS model

Two thousand three hundred statements are used to model this part which, consisting of the main controller, the communication processor and the data access processor, is the most important part in the model.

(1) Main controller—This component manages the whole DBMS, which means that it gathers log information, controls the status of the data base system, schedules messages and communicates with and controls messages. It gives control to the communication processor when an input message arrives. After communication processor's service, it schedules the message according to the priority and activates an application program. A data access call from an application program makes it give control to the data access.

(2) Communication processor—It services input and output messages under communication support in the operating system and gets control when, (a) communication support completes message input and output, and (b) a message to send exists. It moves an input message from the input message queue to the scheduling queue and makes a copy on a disk at the same time. Transmission completion service of an output message causes it to issue a request to restart polling to communication support. When a message is in the output message queue, it issues a command to send out the message to communication support.

(3) Data access—It is activated by a data access call from an application program. Its major modules are service routines for data access calls, i.e., retrieve, insert, replace and delete, buffer and disk space management modules, and modules to support access methods (SAM, ISAM, DAM). The model supports a set of primitive data access call functions including Get Random, Get Random Next, Get Immediate Next, Get Link, Insert Random, Insert Immediate Next, Replace and Delete. For example, Get Random selects a record randomly among all data base records given by data base definition, determines the physical address, and accesses it; Get Random Next calculates the distance, within a record, between the segment

TABLE II—File Definition Table

No.	CONTENTS
1	Access Method
2	Start Address in Disk Number
3	Start Address in Cylinder Number
4	Number of Blocks Allocated
5	Block Length
6	Number of Blocks per Track
7	Logical Record Length

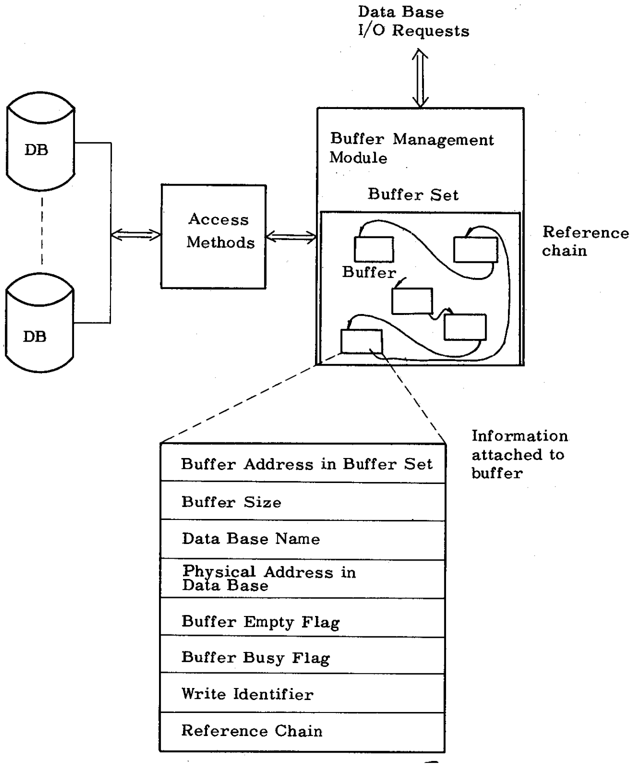


Figure 3—Buffer management position and buffer information

to be obtained and the one accessed before based on the hierarchical structure of the data base and each segment's occurrence and gets it; Get Immediate Next brings the next segment immediately after the current segment; Get Link issues one random physical read to the specified data base.

The LRU (Least Recently Used) method is used in the buffer management module which is one of the most important modules in the DBMS model. All data base I/O requests are processed through this module. Figure 3 illustrates the position of buffer management and information attached to each buffer for its manipulation of the buffer set. Buffers in the buffer set are chained by reference chain. At the time of data base access the buffer management module searches the buffer set at first by tracing this chain, and if the required data don't exist in the buffer set it tries to get buffer space to read the data from disk storage by tracing the chain backwards. The module also controls concurrent data base access.

Application program model

User application programs are modeled in this part. The model mainly involves CPU steps spent in application programs, control of the process and data access calls with the following format:

DA call function, data base name, segment name

Example:

```
    Process     200 (instruction steps)
    DA          GR, DB1, SG1 (GR: Get Random)
    CTR=5
A   Process     100
    DA          GRN, DB1, SG2 (GIN: Get
                Immediate Next)
    CTR=CTR-1
    If          CTR≠0 Go To A
```

EXAMPLES OF SIMULATION

Hardware configuration—Insurance company

Figure 4 shows all equipment components, some of which (the console typewriter, card readers, line printers and most magnetic tapes) are not necessary for simulation.

The central processing unit (CPU) has a 0.1μ second machine cycle time, a 0.9μ second memory cycle time and 16K byte high speed buffer memory and supports four-way interleave.

We decided on a CPU speed of 1.6μ seconds per instruction which was gotten by the other project.

There exists no channel interference as the I/O processor processes all data I/Os.

- Forty (40) disk storage devices are connected with 6 control units and 2 block multiplexor channels and have 30 millisecond average seek time, 16.7 millisecond cycle time, a data transfer rate of 800K bytes per second, and a 100 million byte capacity per spindle. The model also simulates RPS (Rotational Position Sensing) action.
- Only one or two magnetic tapes for logging are necessary to simulate the system because simultaneous background jobs are left out of consideration. The tapes have a data transfer rate of 240K bytes per second and 3 millisecond start time.
- There are 250 terminals connected with 30 lines with a speed of 1200 bits per second. Message length is within a range of 30 to 800 bytes. Poisson arrival of messages is assumed.

Data bases—Insurance company

Eleven data bases, most of them constructed by ISAM and DAM combinations, are contained in the system and have totally about 6 million records which occupy 32 disk packs. The data structures are not so complex, however, as their hierarchical levels are two or three. The largest data base is the car insurance data base which has 1.3 million records and requires 9 disk packs. ISAM files have four kinds of indexes, super index, master index, cylinder index and block index, where super and master indexes are always in core and also cylinder indexes in some cases. All data bases have the same block size of 2400 bytes which allows for five blocks per track.

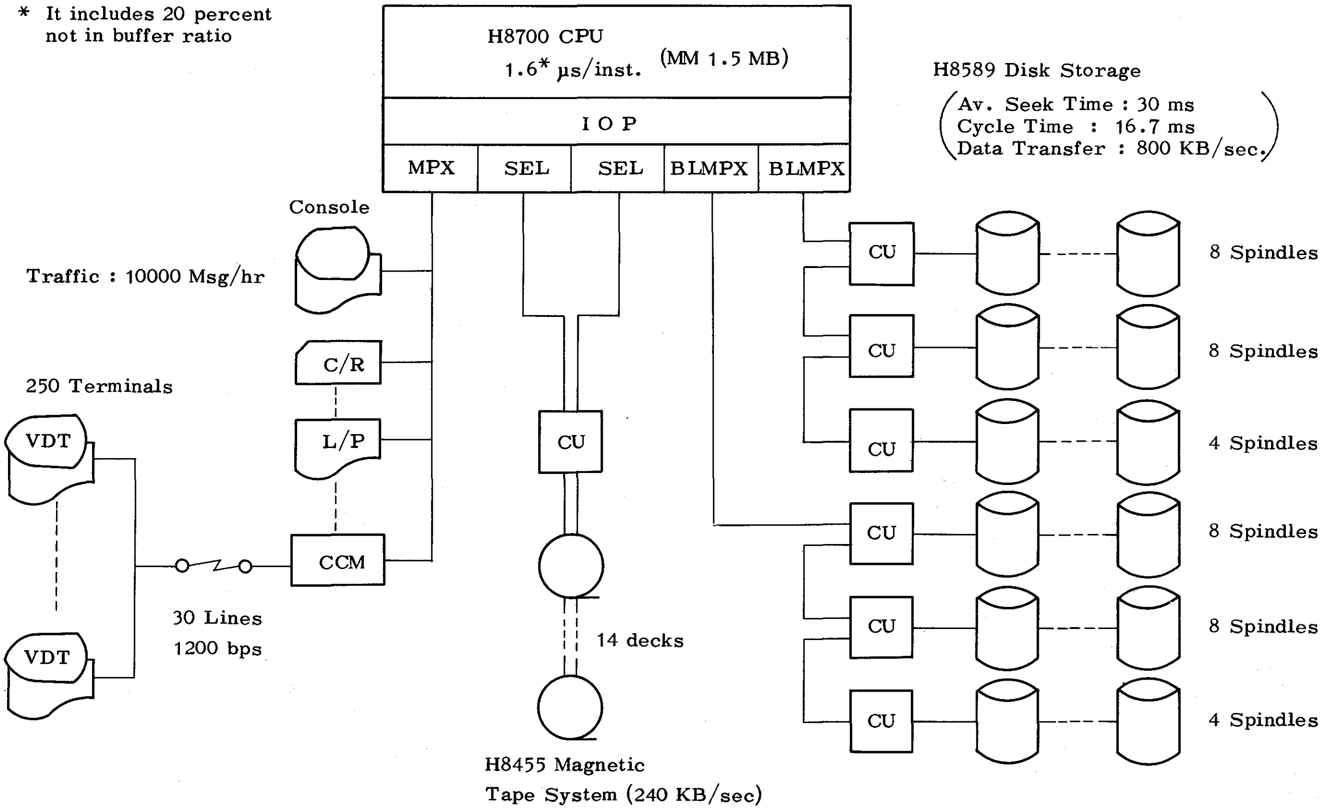


Figure 4—Hardware configuration

Application programs—Insurance company

The system employs 60 different application programs which can be classified into four categories. The first covers inquiry services, the second is registration of a new contract and its document issue, the third is continuance of a contract, and the last is the update of a contract. All categories but the first one require data base updates. New contract processing takes a particularly long time on account of many data base access calls (more than ten get, insert and replace calls) to four or five data bases. Five partitions where application programs run are reserved in the model, which means that a maximum of five application programs are simultaneously executed.

Results—Insurance company

We simulated 15 production runs with different message traffic, message scheduling strategies (concurrent DB access checks), application programs and number of partitions for application programs. Three results selected among those provided by these runs are shown in Table III. Result 1 based on the system environment explained above tells that the system failed to process the traffic of 10000 messages per hour (response time, its standard distribution and current contents of scheduling queue). The causes are neither hardware equipment bottlenecks (CPU, channel and disk utilizations) nor a task bottleneck (number of concurrent

TABLE III—Simulation Results

	1	2	3
Message Traffic (Msg./hr.)	9500	9600	15200
Average Response Time (sec.)	7.8	1.0	1.6
Standard Deviation of Response Time (sec.)	19.9	0.84	2.1
CPU Utilization (%)	31.0	26.0	38.6
Max. Channel Utilization (%)	5.6	5.1	7.5
Max. Disk Utilization (%)	28.5	26.4	40.0
Average Number of DB I/Os (/Trans.)	10.5	6.2	5.2
Current Contents of Scheduling Queue (Trans.)	131	1	0
Average Number of Concurrent AP Tasks	2.2	1.8	2.5
Clock Value in the Model (sec.)	438	573	393
Simulation Time (min.)	220	220	220

AP tasks) because there can exist five application program tasks at a time. The real bottleneck lay in the area of concurrent data base updates. All application programs except inquiry services issue data base update calls and if they are going to update the same data bases, only the message which has the highest priority is scheduled, while other messages remain in the scheduling queue even if all other partitions are empty. In this case, application programs of three categories update the same data bases, and the new contract processing has the highest priority and the heaviest load of all as mentioned before and accounts for a quarter of the message traffic, which means that messages belonging to the other two categories can hardly be scheduled. Actually we found, by checking the scheduling queue, that these messages had waited over 70 seconds there when the simulation stopped. We suggested to the system designers to eliminate on-line updates in the new contract processing and to maintain data bases in batch mode or batch message processing mode. Result 2 shows how the new contract processing without data base updates improved the response time. There is no performance problem at all in Result 2. Result 3 tells that the modified system can easily bear the traffic of 15000 messages per hour.

Other results

The following simulation results were obtained in a different system environment, which will not be explained in detail here.

Figure 5 shows the relationship between data base buffer set sizes and the response time with message traffic as a parameter. The number of concurrent application program tasks is five, the block size of data bases is 3500 bytes, and application programs retrieve a record randomly among all data base records. Figure 5 tells that

- buffer set size hardly affects the response time with low message traffic,
- more buffers are required as message traffic goes up, buffer requirement becomes saturated, however, and further buffers don't shorten the response time.

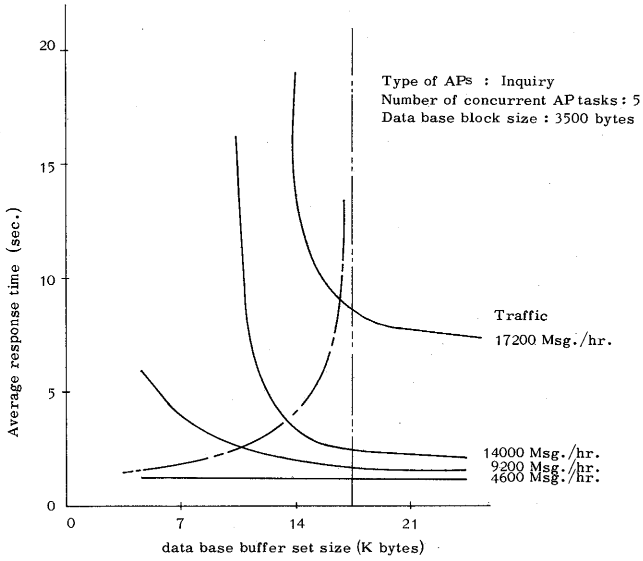


Figure 5—DB buffer set size vs. response time

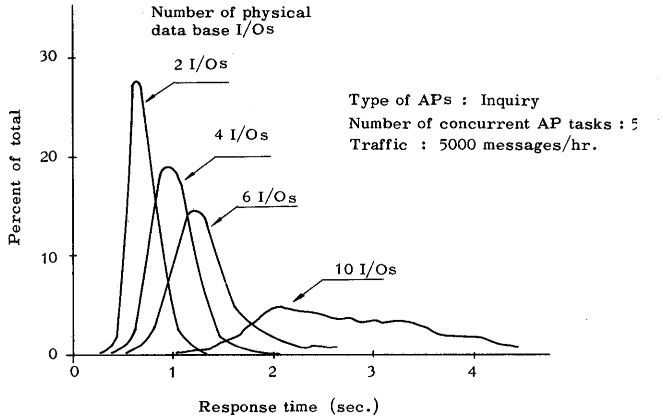


Figure 6—Response time distribution (1)

The dotted line in the figure shows the minimum buffer set size required at each value of message traffic and we can see the necessary and sufficient buffer set size to contribute to the system throughput is 17500 bytes which can contain five data base blocks, the same number as concurrent application program tasks.

This is caused by the facts that a required record seldom exists in the buffer set because of random retrieval and the retrieval process needs one buffer at a time.

Response time distribution with the number of data base I/Os as a parameter is plotted in Figure 6. The effect of the number of data base I/Os on response time distribution is dramatic. Figure 7 shows the response time distribution with physical block lengths of data bases, as a parameter, comparing the same block length with two kinds of block lengths in

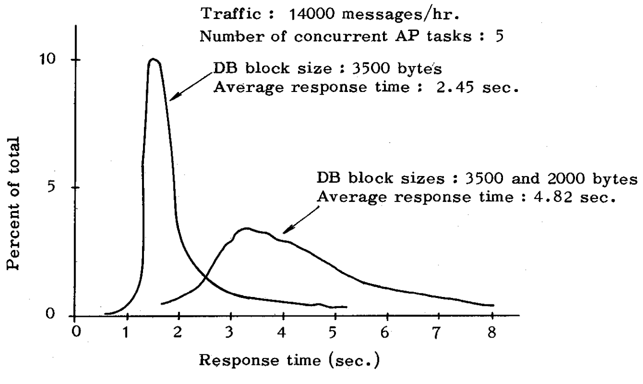


Figure 7—Response time distribution (2)

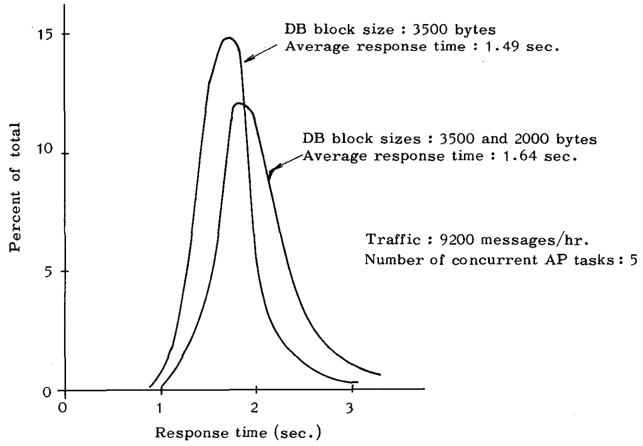


Figure 8—Response time distribution (3)

the same system environment. Average response time in the case of two kinds of block sizes becomes twice the one in case of the same block size, and the same trend exists for the deviation. The reason is that the buffer management module has the common buffer set for all data bases and manages it dynamically. More than one block size causes core fragmentation and frequent requirements for garbage collection, which in turn increases the internal task wait time in the buffer management module.

The effect becomes remarkable as the number of requests for buffer management increases with higher message traffic and/or frequency of data base I/O requests per meassage. Figure 8 which curves were obtained under the same environment as Figure 7 except message traffic demonstrates this. We can see from the figure that there is no big difference between the two results when the message traffic is 9200 messages per hour.

CONCLUSION

A simulation model which uses an event-driven type simulator was developed for the purpose of performance prediction of data base systems and research in the field of data base management system architecture. Several areas in which the model has been and is currently being used are as follows.

- Performance evaluation of an on-line design information system of a manufacturing company
- Evaluation of design alternatives of an on-line data base system in an insurance company
- Simulation of buffer management characteristics
- Simulation of concurrent processing
- Memory hierarchy of data bases (main memory—drum or fixed head disks—movable head disks) vs. system throughput

We have collected and will collect the following data for tuning of the model.

(a) Measurement of CPU dynamic steps and module activity by an instruction tracer.
(b) Measurement of CPU, channels and disk storage devices activities by a hardware monitor. It includes data as described below in batch and on-line environment.

- CPU active time, CPU active and supervisor state time
- Process steps, process steps in supervisor state
- Number of supervisor calls issued
- Channel busy time
- Disk seek time and count, search/read time and count, write time and count, RPS time and count.

We finished measuring them, finished analyzing the results in batch environment and are currently analyzing the results in on-line environment. The model is reasonably accurate in batch environment.

As this model uses an event-driven type simulator, real time/simulated time ratio is generally high (i.e., simulation cost is high). The value of that ratio depends upon the density of events (message traffic, number of data base access calls, etc.). From our experiment, the value is between 20 and 100.

ACKNOWLEDGMENTS

The authors wish to express their grateful thanks to Mr. S. Mimura and Mr. H. Sakai of the Software Works for providing us the opportunity of this research. We also would like to thank Mr. K. Arai and Mr. Y. Kasai for their suggestion and support concerning the simulation package.

REFERENCES

1. Senko, M. E., E. B. Altman, M. M. Astrahan, and P. L. Fehder, "Data Structures and Accessing in Data-Base Systems," *IBM Systems Journal*, No. 1, 1973.
2. Collmeyer, A. J. and J. E. Shemer, "Analysis of Retrieval Performance for Selected File Organization Techniques," *Proceedings of the Fall Joint Computer Conference*, 1970.
3. Stimler, S. and K. A. Brons, "A Methodology for Calculating and Optimizing Real-Time System Performance," *Comm. of the ACM*, Vol. 11, No. 7, 1968.
4. Lum, V. Y., H. Ling, and M. E. Senko, "Analysis of a Complex Data Management Access Method by Simulation Modeling," *Proceedings of the Fall Joint Computer Conference*, 1970.

Weight-balanced trees*

by J. L. BAER
University of Washington
Seattle, Washington

INTRODUCTION

It is now recognized that binary search trees are structures which can be used efficiently for the organization of files and directories. The ease of insertion and deletion of nodes makes trees very appealing for directories which are often modified. By comparison with a sequential table organization, some additional memory is required for the links between nodes. From a cost-effective viewpoint, this is generally more than compensated for by the savings in searching (for a linear table) and inserting (for an ordered table).

A number of studies have been conducted in order to build trees with as small an average searching time as possible while keeping to a reasonable degree the amount of computation and of extra memory needed for the insertion algorithms. Depending on the type of application, these studies have fallen into two categories: those concerned with trees where the nodes have a uniform weight as exemplified by AVL trees,[3,4] binary B-trees,[9] and BB or bounded balanced trees;[6] and those which consider weighted trees, the weight being for example a frequency of query.[2,4,5,8]

Our primary intention was to concentrate exclusively on weighted trees for the following reasons. First, they are more representative of common directories; second, dynamic self-optimization of weighted trees has not been considered yet; and finally, it was felt that the techniques used in non-weighted trees could be easily modified to be applied efficiently to weighted trees. Hence, we present two closely related algorithms, called weight-balanced (WB), for constructing dynamically self-optimizing binary search trees. Results of simulation experiments show that near-optimal trees are obtained.

But, if the algorithms are applied to non-weighted trees, we obtain results in the same range as those yielded by AVL and BB algorithms. This is not surprising since WB algorithms produce BB(1/4) trees in the worst case. A discussion of the relative advantages and disadvantages of the various methods is then in order. It will show the flexibility of the WB approach.

DEFINITIONS AND RELATED WORK

A binary tree is either the empty tree T_0, or the triple (T_L, r, T_R) where T_L and T_R are the *left* and *right binary subtrees* and r is a special node called the *root*. Given a node P in the tree and the subtree $(T_L{}^P, P, T_R{}^P)$, its *left son* is the root of $T_L{}^P$ and its *right son* the root of $T_R{}^P$. Conversely, each node (but the root) has a *father* for which it is a son. A node without a son is a *leaf*. Each node in the tree contains a key. A binary tree is a *binary search tree* (abbreviated b.s.t.) if for all nodes P, the keys of the left (right) subtree are less (greater) than the key of P. In a *weighted binary search tree* (w.b.s.t.) each node has a *weight* associated with it. The *level* of a node P in a tree is one if P is the root, or else the level of its father plus one. The *height* of a node is one if it is a leaf, or else the maximum height of its sons plus one.

In addition, we define the *total* of a node as its weight if it is a leaf, or otherwise as the sum of the total(s) of its son(s) and its own weight.

The weighted *path length* of a w.b.s.t. T of n nodes is

$$\text{Path}(T) = \sum_{i=1}^{n} \text{level } (i) \cdot \text{weight } (i)$$

In terms of computer representation, each node P will contain four fields; namely LLINK(P), INFO(P), TOTAL (P), and RLINK(P). LLINK(P) points to the left son of P or equals λ if there is no left son and likewise for RLINK(P); INFO(P) is the key, and TOTAL(P) is as defined previously.

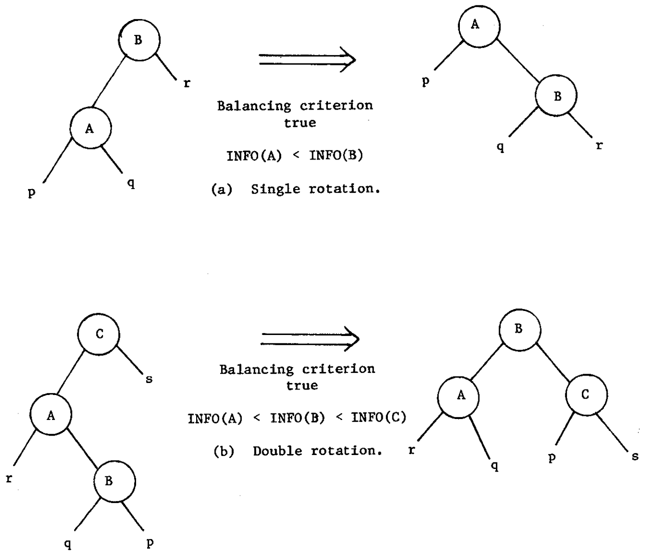
Given a set of n keys and associated weights, there exists an algorithm[4] which produces the optimal w.b.s.t. that is the tree with minimal path length. However, this algorithm presents two disadvantages:

(1) It requires a processing time of the order $0(n^2)$ and, most importantly, additional memory requirements of $0(n^2)$ units. Thus it becomes rapidly impractical when the number of nodes increases.
(2) The weights of all nodes must be known in advance.

Circumventing these difficulties is possible if one is ready to settle for near-optimal trees instead of complete optimality. Heuristic methods, approaching the optimal solution within 2 or 3 percent, have been devised.[2,5,8] The algorithms run in time between $0(n \log n)$ and $0(n^2)$ depending on the weight distribution and require only $0(n)$ memory units. Yet, the second assumption above is still enforced.

We introduce next new algorithms which will yield also near-optimal trees with a similar requirement of $0(n)$ mem-

* This work was supported by NSF Grant GJ-41164.



(a) Single rotation.

(b) Double rotation.

Figure 1

ory units. The main advantage of our scheme is that assumption (2) above is no longer necessary. That is, the algorithms work on evolving structures corresponding to dynamic queries in a directory and "balance" the tree accordingly. In the case where all weights are known in advance, its running time is of the same order as those of the already mentioned algorithms for comparable results. This is not surprising since many heuristics lead to near-optimal trees.[5]

The weight-balanced algorithms that we introduce are adaptations of techniques used in the AVL and BB trees to the case of w.b.s.t.'s. Recall that an AVL tree is a (non-weighted) tree such that the height of the left son of any node never differs by more than one from the height of the right son. The algorithm to build an AVL tree consists of a top-down search to find the place of insertion followed by a bottom-up traversal of the search path with appropriate "balancing" of the tree. The data structure used is similar to the one described above with the only difference being that the TOTAL field is replaced by a smaller (2 bits only) BALANCE indicator.

BB trees of balance α, are such that at every node P:

$$\alpha \leq \frac{\text{TOTAL(LLINK(P))}+1}{\text{TOTAL(P)}+1} \leq 1-\alpha, \quad 0<\alpha\leq \tfrac{1}{2}$$

A pure top-down algorithm to insert an item in a BB(α) tree for any α in the range $0 \leq \alpha \leq 1-\sqrt{2}/2$ is known.[6] As in the AVL trees, the balancings are of the two types shown in Figure 1. The data structure is the same as the one already presented.

SELF-OPTIMIZING WEIGHTED BINARY SEARCH TREES

Self-optimizing weighted structures have been investigated only in the case of linear lists without frequency counters under the assumption that the extra cost incurred by the space reserved for the counters is not effective. Two heuristics have been tested with success: "move up front", which moves the queried entry at the head of the list, and "transposition", which moves it up one notch unconditionally. The latter performs always as well as the former,[7] and in addition, can also be easily implemented with a sequential allocation technique.

In the case of b.s.t.'s, dynamic optimality is impractical for the reasons mentioned previously. Direct extensions of the above heuristics imply restructuring of the tree at every insertion with no assurance of good results. In fact, one can construct realistic cases giving a worst case behavior. Therefore we need a more selective criterion for balancing.

The heuristic that we propose has for its goal to minimize locally the weighted path length on the path of insertion. More specifically, turning our attention to Figure 1, with p, q, r and s being the TOTAL's of the respective sons of A, B and C:

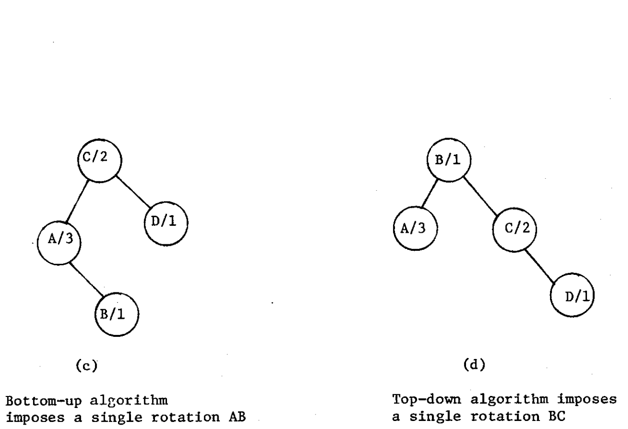
A single rotation will be performed if (Figure 1a):

$$\text{weight(A)}+p>\text{weight(B)}+r$$
$$(\text{or}\quad \text{TOTAL(A)}-q>\text{TOTAL(B)}-\text{TOTAL(A)}),$$

thus minimizing locally the path length of the subtree of new root A.

A double rotation will be performed, after checking that a single rotation did not apply, if (Figure 1b):

$$2\ \text{weight(B)}+p+q>\text{weight(C)}+s$$
$$(\text{or}\ 2.\text{TOTAL(B)}-p-q>\text{TOTAL(C)}-\text{TOTAL(A)})$$



(a)
original tree

(b)
Tree without rebalancing after insertion of A/3

(c)
Bottom-up algorithm imposes a single rotation AB

(d)
Top-down algorithm imposes a single rotation BC

Figure 2

thus minimizing again locally the path length of the subtree of new root B. Hence we define as *weight-balanced trees* (WB trees) those trees obtained dynamically by the application of the above balancing criterion.

The algorithms that we introduce are modelled after the bottom-up AVL construction on one hand and after the top-down BB algorithm on the other hand. (Computer programs can be obtained from the author.) Both can be used with incremental weights corresponding to individual queries or with total weights (or fractions of them) for the periodic rebalancing of directories.

Intuitively it appears that the bottom-up algorithm having more information about weights on the insertion path should perform slightly better. For example, if the node A of weight 3 were introduced at the left of B in the tree of Figure 2a, then the bottom-up algorithm would yield the tree of Figure 2c, and the top-down algorithm the tree of Figure 2d resulting in an advantage of one unit for the former. However, as seen later, the performance of the bottom-up algorithm is only marginally better. Yet, it should not be discarded despite its extra space or pointer manipulation requirement since it can sometimes be used more conveniently as seen in the next section.

Analytical results on w.b.s.t.'s are scarce.[5,10] While using heuristics care must be taken in their development since some will tend asymptotically to construct random trees (e.g., insert in order of decreasing weights). The difference between random trees and optimal trees being on the order of 40 percent on the average, one can understand why a category of heuristics yielding "balanced" trees is needed. By examining the results of some simulation experiments, we can safely assume that the one we propose falls into that latter class.

TABLE I—Bottom-up Algorithm on Weighted Trees.

	Tree 1	Tree 2	Tree 3
1. Optimal	3.437	2.992	5.200
2. Random	4.804 (43%)	3.820 (28%)	6.074 (17%)
3. Decreasing weights	4.042 (18%)	3.142 (8%)	5.726 (10%)
4. Decreasing weights and rebalancing	3.437 (–)	2.992 (–)	5.212 (.2%)
5. Self-optimizing 100%	3.523 (2.2%) 88 rotations	3.007 (.5%) 48 rotations	5.241 (.8%) 31 rotations
6. Self-optimizing 10%	3.526 (2.4%) 61 rotations	3.059 (2.2%) 21 rotations	5.314 (2.2%) 20 rotations
7. Self-optimizing 1%	3.550 (3.3%) 35 rotations	—	—

TABLE II—Top-down Algorithm on Weighted Trees.

	Tree 1	Tree 2	Tree 3
1. Optimal	3.437	2.992	5.200
2. Random	4.804 (43%)	3.820 (28%)	6.074 (17%)
3. Decreasing weights	4.042 (18%)	3.142 (8%)	5.726 (10%)
4. Decreasing weights and rebalancing	3.502(2%)	2.992 (–)	5.238 (.7%)
5. Self-optimizing 100%	3.563 (3.6%) 35 rotations	3.060 (2%) 27 rotations	5.265 (1.2%) 26 rotations
6. Self-optimizing 10%	3.569 (3.8%) 29 rotations	3.110 (4%) 17 rotations	5.337 (2.6%) 19 rotations
7. Self-optimizing 1%	3.575 (4%) 23 rotations	—	—

Simulation results

In order to test our hypothesis we performed a series of experiments on the following three trees:

Tree 1: 31 nodes; total queries$\simeq$85000.
Tree 2: 35 nodes; total queries$\simeq$1300.
Tree 3: 27 leaves carried the information (i.e., a tree of 53 nodes); total queries = 1000.

The results of the simulation are shown in Table I for the bottom-up algorithm and Table II for the top-down. Ten sample runs were used for lines 2, 5, 6 and 7. Lines 1 through 4 show the average search times in the optimal (1) and random (2) cases, with insertion of keys (total weight at once) in decreasing weight order without balancing (3), and with balancing (4). In lines 5 to 7 each query is treated individually and is selected randomly assuming a uniform distribution on the remaining queries. Balancing is performed on either all queries (5), the first 10 percent (6) or the first 1 percent (7).

From this series of experiments we observe that:

(1) The bottom-up algorithm performs slightly better than the top-down.
(2) The random trees are (on the average) between 17 and 43 percent worse than the optimal, and the "decreasing weight" heuristic is not good, thus confirming the theoretical analysis.[5]
(3) The "decreasing weight with balancing" performs extremely well (within 2 percent of optimality).
(4) The self-optimizing feature is very efficient. Near-optimal trees are obtained with a small number of rebalancings. Under our assumption of uniform distribution of queries, this is quite encouraging, since it

implies that balancing could be performed only at the creation of the directory on each individual query and then periodically later on.

These simulation results show that WB trees lead to efficient heuristics, yielding near optimal trees. The amount of space needed is minimal and the extra computation time not significant if one limits the restructuring to the initial creation time of the directory.

BALANCING NON-WEIGHTED TREES

The algorithms presented in the previous section were not intended to be used for non-weighted trees. Yet, when applied to those structures they yield results quite comparable in efficiency and flexibility with those obtained through the AVL and BB techniques.

Worst case behavior

It is known that Fibonacci trees are the worst case of AVL trees, i.e., the highest level is bounded by $1.44\log_2(n+1)$ for a tree of n nodes. In the case of BB (α) trees, it has been shown[6] that the worst case is $(\log_2(n+1)-1)/\log_2(1/(1-\alpha))$.

In order to obtain a bound on the worst case for WB trees, we show that they are BB(1/4). The following lemma is true for WB trees constructed either by the top-down or bottom-up algorithms. The proofs assume a bottom-up algorithm. The method is to show that some trees cannot arise when the WB algorithms are executed. Because of space limitations we show only part of the proof in detail.

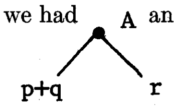
Lemma: In a WB tree each node P is such that $|T_L^P| \le 3|T_R^P|+1$, and $|T_R^P| \le 3|T_L^P|+1$.

Proof: (By induction on the number of nodes n in the tree.)

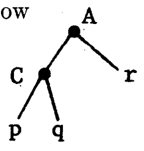
The lemma is true for $n=1$, 2, 3 as can be easily seen by construction. Assume it is true for n (hypothesis H1). Let us insert a node in the tree and show that the WB algorithm results in a tree with H1 verified for $n+1$. We only have to check on the path of insertion that the lemma's property is conserved. Let k' be the height of the node C on the path of insertion. The lemma's property is still true for $k'=1$, 2 as can be shown directly by construction. Assume that it is true for $k'=k-1$ (H2). We show that it is still true for $k'=k$. We distinguish 3 cases:

A. **No rotation**

Before insertion we had

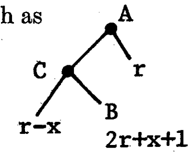


and we have now

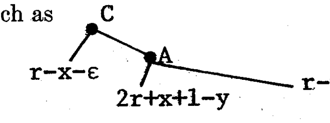


The only difficulty is in showing that $p+q+1\le 3r+1$, or $q\le 3r-p$ when $p<q$. Assume $p=r-x$ and let us show that $q=2r+x+1$ is impossible ($q>2r+x+1$ contradicts H1).

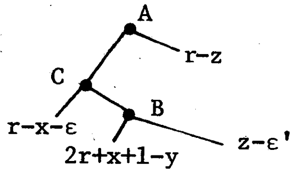
Thus we have to show that a subtree such as



cannot result when a key has been inserted at the left of A. Such a tree cannot be constructed directly since at most $2r+1$ keys can be inserted at the left of A without implying a rotation on A. The latest rotation on A could not have been a single rotation AC since it would have implied a subtree such as



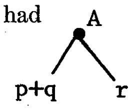
and insert which implies $2\epsilon+3x+1\le y$, i.e., at least $1\le y$, because of the induction H1. But then the first insertion of the y remaining keys greater than INFO(C) and less than INFO(A) would rotate again A and C. Finally, a double rotation such as



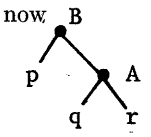
would fail for the same reasons.

B. **Single rotation**

Before insertion we had



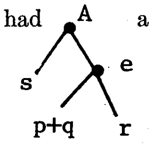
and we have now



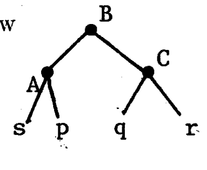
The (easy) proof stems from H1 and a possible contradiction on H2.

C. **Double rotation**

Before insertion we had

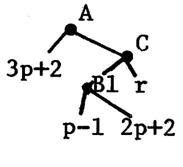


and now

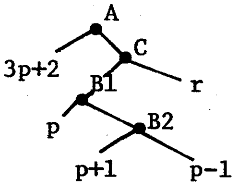


The difficult part of the proof is to show that $s\le 3p+1$ (or $r\le 3q+1$). We use the same contradictory approach in assuming $s=3p+2$. Then before the rotation we had:

either



or



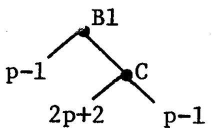
(next insertion on left of B1) (next insertion on right of B2)

TABLE III—AVL, BB and WB Algorithms

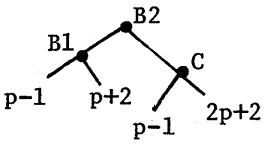
# of nodes	Average searching time					
	opt	AVL	BB [1 $-\sqrt{2}/2$]	WB (top down)	WB (bottom up)	Random
1000	8.98	9.207 (.46)	9.267 (.40)	9.161 (.51)	9.155 (.54)	11.981
750	8.66	8.790 (.47)	8.816 (.40)	8.740 (.51)	8.741 (.52)	11.157
500	7.99	8.193 (.47)	8.216 (.40)	8.157 (.51)	8.150 (.52)	10.665

We show that obtention of the subtrees of root B1 is not possible. Evidently this is true for $p=1$. Assume it is not possible for $p-1$ (hypothesis H3). We prove then that it is not possible for p. We detail the proof only for the first subtree.

The last insertion to obtain the subtree of root B1 cannot come from the right of C since we would have had either

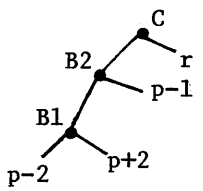


or

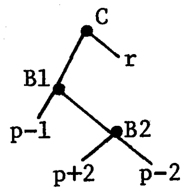


(forbidden by H1) (forbidden by H3)

Evidently H3 forbids that the latest insertion comes from the left of B1. The only two situations left open are:



or



This construction can be carried on further, but we can see that the $p+2$ keys K such that $\mathrm{INFO(B1)} < K < \mathrm{INFO(B2)}$ are never introduced. Therefore, either of H1 or H3 will soon be contradicted, thus showing the impossibility that $s=3p+2$.

Theorem: The maximum level of a WB tree is $0(\log_2 n)$ where n is the number of nodes in the tree.

Proof: The lemma shows that

$$\tfrac{1}{4} \leq \frac{\mathrm{TOTAL(LLINK(P))}+1}{\mathrm{TOTAL(P)}+1} \leq \tfrac{3}{4}$$

i.e., WB trees are BB(1/4); the highest level is hence[6]$\sim$2.32 $(\log_2(n+1)-1)$, or $0(\log_2 n)$.

Average searching time

As for all b.s.t.'s the average searching time in a WB tree is $0(\log_2 n)$. To evaluate the efficiency of the WB algorithms, we performed the following experiment. Sets of 500, 750 and 1000 different keys were generated randomly and we applied the AVL, BB$(1-\sqrt{2}/2)$ and WB algorithms for their insertions in a binary search tree. The averages of 25 sample runs are shown in Table III. As can be seen, the WB trees are slightly more performant (.5 percent to 1 percent), but certainly not significantly. The average number of rotations/insertion is also indicated (figures in parentheses). The WB algorithms require between 5 percent and 10 percent more rotations than the other two methods.

This larger number of rotations is compensated by the following advantages. First, in comparing with the BB algorithm, there is no need for division which is a time consuming operation. Second, in comparing with the AVL technique, we can use a top-down algorithm so that we do not have to go back up the path. Finally, the fact that we can choose between two algorithms allows us to face easily

TABLE IV—AVL, BB and WB Flexibility (1000 nodes)

	$\delta = 1$ $\alpha = 1 - \sqrt{2}/2$ $\gamma = 0$	$\delta = 2$ $\alpha = .25$ $\gamma = 1$	$\delta = 3$ $\alpha = .20$ $\gamma = 2$	$\delta = 4$ $\gamma = 3$	$\delta = 5$ $\gamma = 4$
AVL	9.207 (.46)	9.416 (.21)	9.665 (.12)	9.925 (.07)	10.23 (0.05)
BB	9.267 (.40)	9.444 (.26)	9.765 (.16)		
WB top down	9.161 (.51)	9.251 (.26)	9.344 (.18)	9.427 (.13)	9.505 (.11)
WB bottom up	9.155 (.54)	9.244 (.27)	9.333 (.18)	9.422 (.13)	9.499 (.11)

the contradictory situations: the key to be inserted is certainly not present, hence use a top-down algorithm; and, the key to be inserted might be present, hence use a bottom-up algorithm and cancel the balancing if the key were present.

Space required

The amount of space required is of the same order for the three algorithms. As observed in Reference 4, the little savings in AVL trees is compensated by the need of the TOTAL field (or equivalent) if one wants to use the structure efficiently as a linear list

Flexibility of the algorithm

It is interesting to see how the algorithms behave if one relaxes the balancing criteria in order to reduce the number of rotations. In terms of AVL trees[3] this is akin to letting the difference in heights be some δ greater than 1 and for BB trees to having α be less than $1-\sqrt{2}/2$. For WB trees this means that we will have a single rotation if

$$\text{weight}(A)+p>\text{weight}(B)+r+\gamma \qquad \text{(Figure 1a)}$$

and a double rotation if

$$2.\text{weight}(B)+p+q>\text{weight}(C)+s+\gamma \qquad \text{(Figure 1b)}$$

The above sets of 1000 keys were used again to construct AVL trees with δ varying from 1 to 5, BB trees with α being $1-\sqrt{2}/2$, 0.25 and 0.20, and WB trees with γ varying from 0 to 4. As before, the number of rotations was monitored. The results summarized in Table IV show that WB algorithms provide a fine tuning of the average search time with decreases in rotations (proportionally) as great as for AVL trees.

At this time it does not seem possible to rate the "goodness" of the different algorithms since there is no evident connection between the parameters α, γ and δ. The only fair assessment that one can make relative to WB trees is that both the worst case and average searching times (for trees of approximately 10^3 nodes) are of the same order of magnitude as those obtained by using AVL and BB trees. Furthermore, the parameter δ can be used very efficiently for tuning the algorithm. Mainly, it should be emphasized that since the WB technique can also be applied to weighted trees and since there is a choice between a top-down and a bottom-up algorithm, WB trees provide a very flexible tool.

CONCLUSION

In this paper we have presented a new technique for balancing search trees in order to minimize the average searching time of a key in the tree. The technique is based on the concept of weight balance and is applicable to both weighted and non-weighted trees. In the former case, near-optimal trees can be obtained dynamically in a self-optimizing fashion. In the latter, the technique gives results of quality comparable to those obtained by AVL or BB methods.

It would be interesting to test further the technique in different environments, as for example paging systems, multiprocessing and in connection with B-trees.[1]

REFERENCES

1. Bayer, R. and E. McCreight, "Organization and Maintenance of Large Ordered Indexes," *Acta Informatica*, pp. 173-189, Volume 1, 1972.
2. Bruno, J. and E. G. Coffman, "Nearly Optimal Binary Search Trees," *IFIP Congress 1971*, pp. 99-103, North-Holland.
3. Foster, C. C., "A Generalization of AVL Trees," *CACM*, 16, 8, pp. 513-517, August 1973.
4. Knuth, D., *Sorting and Searching*, Addison-Wesley, 1973.
5. Nievergelt, J. and C. K. Wong, "On Binary Search Trees," *IFIP Congress 1971*, pp. 91-98, North-Holland.
6. Nievergelt, J. and E. M. Reingold, "Binary Search Trees of Bounded Balance," *SIAM J. Computing*, 2, 1, pp. 33-43, March 1973.
7. Rivest, R., "On Self-Organizing Sequential Search Heuristics," *15th Annual Symposium on Switching and Automata Theory*, pp. 122-126, October 1974.
8. Walker, W. A. and C. C. Gottlieb, "A Top-Down Algorithm for Constructing Nearly-Optimal Lexicographic Trees," *Graph Theory and Computing*, R. C. Read Ed., Academic Press, pp. 302-323, 1972.
9. Bayer, R., "Symmetric Binary B-Trees: Data Structure and Maintenance Algorithms," *Acta Informatica*, pp. 290-306, Volume 1, 1972.
10. Rissanen, J., "Bounds for Weight Balanced Trees," *IBM Journal of R & D*, pp. 101-106, 17, 1, March 1973.

Optimizing distributed data bases—A framework for research*

by K. DAN LEVIN and HOWARD LEE MORGAN
The Wharton School
Philadelphia, Pennsylvania

INTRODUCTION

The interaction between computers and communication technology has steadily developed in recent years bringing with it the creation of large computer communication complexes. Earlier computer communication systems were generally focused around a single large computer installation. Although a strong argument can still be made for serving distributed users with a centralized system, we now witness an increasing number of systems in which information processing and storage functions are distributed among several computers. The idea is to distribute the resources (programs, data, computing hardware, etc.) of each computer site to a widely spread community of users. Various factors may favor a shift toward computer networks, especially with similarly structured organizations which are independent but can be motivated to share. The most well-known example is the network developed under the sponsorship of the Advanced Research Project Agency (ARPA), that links independent research organizations, all involved in solving closely related problems. The libraries of specialized software and data at each research center, make it attractive to link the computers of these organizations. Clearly, access to specialized data bases is a major feature of the networks and the cost incurred by such an access is an important consideration in determining the economic viability of this kind of resource sharing.

Much research is being focused on the problem of minimizing the operating cost of a distributed data base shared by a community of users interconnected through a computer communication network. We ourselves have been studying the problems of finding the optimal file/program locations in computer networks. In this paper, we attempt to provide a framework in which to view this research, to point out those areas which we and others have already studied, and to highlight areas which would be fruitful for others to examine.

A THREE-DIMENSIONAL FRAMEWORK

We can break up the problem of organizing distributed data bases in computer networks along three dimensions:

(1) The level of sharing
(2) The behavior of access patterns
(3) The level of information on the behavior of access patterns.

In the first dimension, the "level of sharing", one may distinguish between the data sharing level (in which independence between programs and files can be assumed) and the "program and data sharing" level, in which dependencies between files and programs exist (as is the case in real heterogeneous computer networks).

When no sharing takes place, there is no allocation problem since each node has to carry a copy of every file and program that might be requested at that node. At the level of data sharing, each node carries a copy of every program but the data files themselves are shared, i.e., the files can be accessed from remote nodes. At this level, the fact that access requests have to be processed by the relevant programs has no effect on the distribution pattern of the requests. Each request can be processed at the node that originated this request. At the highest level, both the programs and the data files are shared and are accessible to other nodes. At this level, program and data sharing, an access request from node A, invokes a program at node B that in turn requires access to file X at node C. Obviously, the distribution of access requests to files is dependent (to some extent) on the location of the programs. Hence, the optimal location of data files is also dependent on the program assignment.

It is true that the distinction between programs and data files is somewhat artificial but one has to make this distinction in a heterogeneous network. While data files can be transferred relatively easily from one computer to another, even where different manufacturers are involved, programs written and compiled under the supervision of one operating system cannot be executed in a different computer. The opportunity costs of "stepping down" from a "sharing" level to "no sharing" are determined by the costs of getting compatible copies of the files at each node. Where data files are involved, the costs of obtaining a compatible copy are determined by the storage cost at the node at which a compatible copy is required. It is much more complicated to obtain an operationally compatible copy of a program. The main cost factor in this case is the cost of conversion from one computer to another, and in most cases the program has to be rewritten. In addition, all versions of the program have to be updated and the problems associated with both

* This research was supported by the Office of Naval Research under Contract N00014-67-A-0216-07.

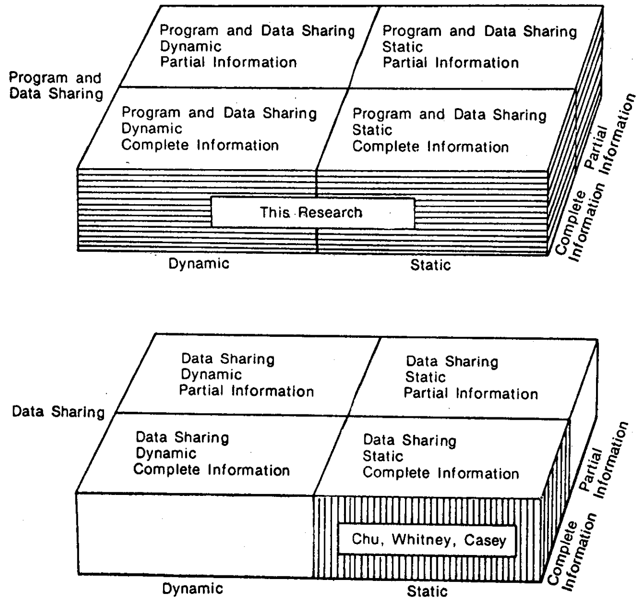

Program and Data Sharing
Program and Data Sharing Dynamic Partial Information
Program and Data Sharing Static Partial Information
Program and Data Sharing Dynamic Complete Information
Program and Data Sharing Static Complete Information
This Research
Dynamic
Static
Partial Information
Complete Information
Data Sharing
Data Sharing Dynamic Partial Information
Data Sharing Static Partial Information
Data Sharing Dynamic Complete Information
Data Sharing Static Complete Information
Chu, Whitney, Casey
Dynamic
Static
Partial Information
Complete Information

Figure 1—Three-dimensional partitioning of the file assignment problem

the initial conversion and the continuous maintenance are well-known. It is most conceivable that in heterogeneous networks, the network members will find it more attractive to share programs than to convert them and by doing so, dependencies between programs and files are introduced into the system.

In the second dimension a distinction should be made between "static" behavior and "dynamic" behavior. "Dynamic" behavior refers to a situation in which the access request patterns are changed over time while in static behavior they remain fixed. Most of real life situations fall under the "dynamic" category.

The level of information available can be dichotomized to "complete" information and partial information. Under "complete" information we assume that the access request patterns are known with certainty, i.e., there are no deviations between the forecasted request patterns and the actual request pattern. When only partial information is available one should expect some disturbances between the forecasted request pattern and the actual request patterns.

Figure 1 illustrates the three-dimensional partioning of the distributed data base problem. Under this framework an interaction between "partial information" and "dynamic" behavior represents access request patterns that are both changed over time and unknown. The evolution of these access requests over time can be regarded as a non-stationary time series.

So far, all the efforts with regard to the optimal location of files in computer networks, were concentrated in the "data sharing" level under static behavior and complete information assumptions (the bottom right-hand box of Figure 1). Two papers by Chu[1] and Casey[2] and a Ph.D. dissertation by Whitney[3] addressed the file location under these restrictive assumptions. As illustrated in Figure 1 our work addresses the file location problem at the "program and data sharing" level, i.e., dependencies between programs and files are taken into account. Clearly, the models developed at this level can handle all the corresponding situations at the lower level just by relaxing the program-file interdependence assumption.

MODEL FORMULATION

The operating costs of the data base are a function of the communication cost to the individual files (incurred by access request from remote locations) and the storage costs of the files in the resident nodes. It is clear that different assignments of files/programs in the network nodes will yield different operating costs. It is also conceivable that in some cases, the maintenance of several copies of the same file (distributed over the network), will be economically attractive. Accordingly, one can construct a model of the operating cost of a file as a function of the number of file copies maintained, the location of these copies and the patterns of access requests to the files. In this model the access request patterns will be regarded as uncontrolled variables, while the number of file copies maintained and their assignments to network nodes will be controlled by the model. In this respect, the focus is on the optimal assignment of data files. We already stated that programs can be executed in a limited subset of the network nodes and since the storage costs of programs are relatively small (as compared to storage costs of data files) the optimization is performed on data files only.

Static file assignment with complete information

A computer network is considered here with N nodes and a data-base constituted of F files and P programs. The problem is to find that distribution of files and programs that minimizes the operating costs of the data base. These costs are formulated below. Every node in the network demands the services of some programs and files. This demand is generated through transactions originated in each node and fall into one of two classes—query traffic and update traffic. The transaction is first routed to its relevant program and from this program a query is transmitted to the nearest file copy while an update message is transmitted to every copy of the file.

Let λ_{ipf} = Query traffic from node i to file f via program p
λ_{ipf}' = Updating traffic from node i to file f via program p
C_{ij} = Communication cost per query unit from i to j
C_{ij}' = Communication cost per update unit from i to j

We differentiate between updates and queries (traffic volume and communication costs), on the premise that queries require faster response time than updating traffic and should be charged accordingly with a higher rate.

Let σ_{jf} = Storage cost of file f at node j
σ_{jp} = Storage cost of program p at node j
α = Expansion factor for query message
β = Expansion factor for update message

The provision of the expansion factors is discussed below.

Let program p and file f reside in node j and node k respectively and assume that a query, x bits length, has been originated at node i. The query is first transmitted to its relevant program p to be processed. The program resides at node j, consequently, x bits are transmitted on the communication link $i-j$. This query is then processed at node j and, as a result, an access request to file f is issued by the program. The form and the length of the access request is different from the original query of x bits. This access request is developed by the program and the host computer software, specifically, it consists of a sequence of commands from the Network Command Language (NCL) and is handled by the Inter Process Communication (IPC) mechanism, see Peebles.[4] Assume the length of the access request is y bits. These y bits are transmitted on the communication link $j-k$ (between the program and the file) and the ratio y/x determines the expansion factor α mentioned above. Similarly, another expansion factor β can be applied to update traffic. Jp denotes the set of nodes at which a given program p can be processed. It is clear that in a homogeneous computer network, or when only data sharing is available, the set Jp for any given p is equivalent to the set of the network nodes.

So far we have defined the uncontrolled variables of the model. We will now define the controlled variables that determine the file locations and the routing discipline.

$$\text{Let } y_{kf} = \begin{cases} 1 & \text{A copy of file } f \text{ is stored at node } k \\ 0 & \text{Otherwise} \end{cases}$$

$$y_{jp}' = \begin{cases} 1 & \text{A copy of program } p \text{ is stored at } j \text{ node} \\ 0 & \text{Otherwise} \end{cases}$$

Where k and j denotes the network nodes' indices, j, $k = 1 \ldots . N$, p denotes the program index, $p = 1 \ldots . P$ and f denotes the file index, $f = 1 \ldots . F$. Recall that p can be processed only in a subset of the network nodes (in Jp).

x_{jkf} indicates that portion of transactions from node j to file f that should be routed to node k. Thus, $0 \leq x_{jkf} \leq 1$ and it can be shown that at the optimum x_{jkf} will assume the values 0 or 1.

$$\text{i.e., } x_{jkf} = \begin{cases} 1 & \text{Transactions from node } j \text{ to file } f \text{ are routed to node } k \\ 0 & \text{Otherwise} \end{cases}$$

$$\text{Similarly } x_{ijp}{}^f = \begin{cases} 1 & \text{Transactions from node } i \text{ to file } f \text{ are routed to node } j \text{ via program p.} \\ 0 & \text{Otherwise} \end{cases}$$

Obviously, one would expect that if $x_{jkf} = 1$, then $y_{kf} = 1$ and if $x_{ijp}{}^f = 1$, then $y_{jp}' = 1$ and $j \in J_p$.

The upper index f (in $x_{ijp}{}^f$) provides us with the possibility that transactions from a given node, that are processed by the same program but require access to different files, can be routed to different nodes at which copies of the program are stored.

We are now in a position to express the traffic flow between nodes in terms of the controlled and uncontrolled variables and to formulate the objective function of the model.

$$\rho_{jf} = \sum_{i,p} \lambda_{ipf} * x_{ijp}{}^f = \text{Query traffic to file } f \text{ processed at node } j$$

$$\psi_{jf} = \sum_{i,p} \lambda_{ipf}' * x_{ijp}{}^f = \text{Updating traffic to file } f \text{ processed at node } j$$

The Model Objective is To Minimize $C =$

$$\begin{aligned}
& \sum_{f,j,i,p} \lambda_{ipf} * C_{ij} * x_{ijp}{}^f &&= \text{Communication cost of queries from initiating nodes to the programs} \\
+ & \sum_{f,j,i,p} \lambda_{ipf}' * C_{ij}' * x_{ijp}{}^f &&= \text{Communication cost of updates from initiating nodes to the programs} \\
+ & \sum_{f,j,k} \rho_{jf} * \alpha C_{jk}' * x_{jkf} &&= \text{Communication cost of queries from programs to files} \\
+ & \sum_{f,j,k} \psi_{jf} * \beta * C_{jk} * y_{kf} &&= \text{Communication cost of updates from programs to files} \\
+ & \sum_{f,k} \sigma_{kf} * y_{kf} &&= \text{Storage cost of files} \\
+ & \sum_{j,p} \sigma_{jp} * y_{jp}' &&= \text{Storage cost of programs}
\end{aligned}$$

Subject to the following constraints:

—To assure the attainment of a feasible solution there must be at least one copy of each file and each program, i.e.,

$$\sum_j y_{jp}' \geq 1 \qquad \surd p = 1 \ldots P$$

$$\sum_k y_{kf} \geq 1 \qquad \surd f = 1 \ldots F$$

—To assure that every transaction to every file via every program and from every node, will have a defined route:

$$\sum_j x_{ijp}{}^f \geq 1 \qquad \surd i = 1 \ldots N, p = 1 \ldots P, f = 1 \ldots F$$

$$\sum_k x_{jkf} \geq 1 \qquad \surd j = 1 \ldots N, f = 1 \ldots F$$

—To assure residency of the appropriate files and programs in accordance with the defined routes:

$$\sum_i x_{ijp}{}^f \leq N * y_{jp}' \qquad \surd j = 1 \ldots N, p = 1 \ldots P, f = 1 \ldots F$$

$$\sum_j x_{jkf} \leq N * y_{kf} \qquad \surd k = 1 \ldots N, f = 1 \ldots F$$

—To assure that program p will reside only in a node at which it can be processed:

$$y_{jp}' = 0 \qquad \surd j \notin J_p, \quad p = 1 \ldots P$$

And y_{jp}', y_{kf}, $x_{ijp}{}^f$, x_{jkf} binary variables.

It has been shown by Levin[5] that the multi-file minimization problem can be decomposed to individual file minimization problems and the cost associated with an arbitrary assignment of an arbitrary file f (Kf) is given by:

$$C(K_f) = \sum_{p\in P_q} \sum_i \lambda_{ipf} * \text{Min}_{j\in J_{P_q}} (C_{ij} + \alpha * \text{Min}_{k\in K_f} C_{jk})$$

$$+ \sum_{p\in P_u} \sum_i \lambda_{ipf}' * \text{Min}_{j\in J_{Pu}} (C_{ij}' + \beta * \sum_{k\in K_f} C_{jk}') + \sum_{k\in K_f} \sigma_{kf}$$

Let $Q(K_f) = \sum_{p\in Pq} \sum_i \lambda_{ipf} * \text{Min}_{j\in J_{P_q}} (C_{ij} + \alpha * \text{Min}_{k\in K_f} C_{jk})$

= Queries cost

$U(K_f) = \sum_{p\in P_u} \sum_i \lambda_{ipf}' * \text{Min}_{j\in J_{Pu}} (C_{ij}' + \beta * \sum_{k\in K_f} C_{jk}')$

= Updates cost

$S(K_f) = \sum_{k\in K_f} \sigma_{kf}$

= Storage cost

So that $C(K_f) = Q(K_f) + U(K_f) + S(K_f)$

The problem is to find that assignment K_f for which $C(K_f)$ is minimum and we will refer to this problem in the section that deals with recent results of our own research.

Dynamic file assignment with complete information

The dynamic file assignment problem involves an optimization process extending over several periods of time. With varying access request rates from period to period, it is conceivable that an optimal assignment at one period is non-optimal at the next period. In this case, an additional cost factor has to be considered and it is not sufficient to invoke the static model (mentioned above) for each pattern of access requests. This factor expresses the transition costs incurred by the file movements from one assignment at a given period to another assignment at the next period. In order to move from one assignment k^1 to another assignment k^2, the files have to be transferred over the communication links from the nodes in K^1 to the nodes in K^2. It follows that in order to find the optimal combination of file assignments over t periods, an aggregate evaluation should be performed.

Let $y_{kt} = \begin{cases} 1 & \text{File copy is assign to node } k \text{ at period } t \\ 0 & \text{Otherwise} \end{cases}$

And let $K_t = \{k \mid y_{kt} = 1\}$ denote an arbitrary assignment of a file at t.

$\mathbf{K}_T = [K_1, K_2, \ldots, K_T]$ An arbitrary arrangement of file assignments at periods 1 to T.

Two major cost factors are associated with each possible combination of file assignment $\mathbf{K}_T$.

(a) Summation over T periods of the operating costs at each period. These costs are given by $C(K_t)$, the cost function of the static model. Recall that $C(K_t)$ is subdivided to three factors, i.e., $C(K_t) = Q(K_t) + U(K_t) + S(K_t)$

(b) Summation over T periods of the transition costs from an assignment in one period to the assignment in the next period. This factor is denoted by $T_t(K_{t-1}, K_t)$.

The transition costs from one period to the next one are determined by the communication costs of transmitting the files over the communication links.

Let Ls denote the length of the file in storage units. And let γ be a transformation factor from storage units to message units.

Then $Lm = Ls*\gamma$ = The number of message units involved in the transmission of the file over the communication links. Transition costs from K_{t-1} to Kt (two arbitrary assignments) are determined by

$$T_t(k_{t-1}, K_t) = \sum_{k(t)\in K_t} \text{Lm} * \text{Min}_{k(t-1)\in K_{t-1}} C_{k(t-1)k(t)}$$

Where $k(t)$ denotes the node index in period t.

The term $\text{Min}_{k(t-1)\,\epsilon K_{t-1}} C_{k(t-1)k(t)}$ reflects the assumption that file transmission from one assignment to the other is carried out in the most economical way.

Thus, the multi-period cost function for an arbitrary combination $\mathbf{K}_T$ is given by:—

$$G(\mathbf{K}_T^*) = \sum_{t=1}^{T} [C(K_t) + T_t(K_{t-1}, K_t)]$$

Implicitly, we assume that the file assignment at t_0 is given. The problem is to find that arrangement of file assignments $\mathbf{K}_T^*$ that will minimize the multi-period cost function.

$$\text{i.e., } G(\mathbf{K}_T^*) = \text{Min}_{\mathbf{K}_T} G(\mathbf{K}_T)$$

where the minimization is performed on all possible combinations of $\mathbf{K}_T$.

File assignment with incomplete information

The main disadvantage of the complete information assumption is that exact measurement of all access rates in the network is necessary. This is seldom the case in practical applications and in this section, we will relax this assumption. We will introduce deviations between the actual demand patterns in the network and the forecasted static access rates, deviations due to the fact that the access rates at each node for each program and file are random variables with underlying probability distributions. The static environment is implied by a "stationary" assumption, i.e., the same probability distribution exists over time for each node, program and file. The dynamic environment is implied by a "non-stationary" assumption. The access request rates are treated as random variables with an underlying probability distribution function. However, unlike the stationary case,

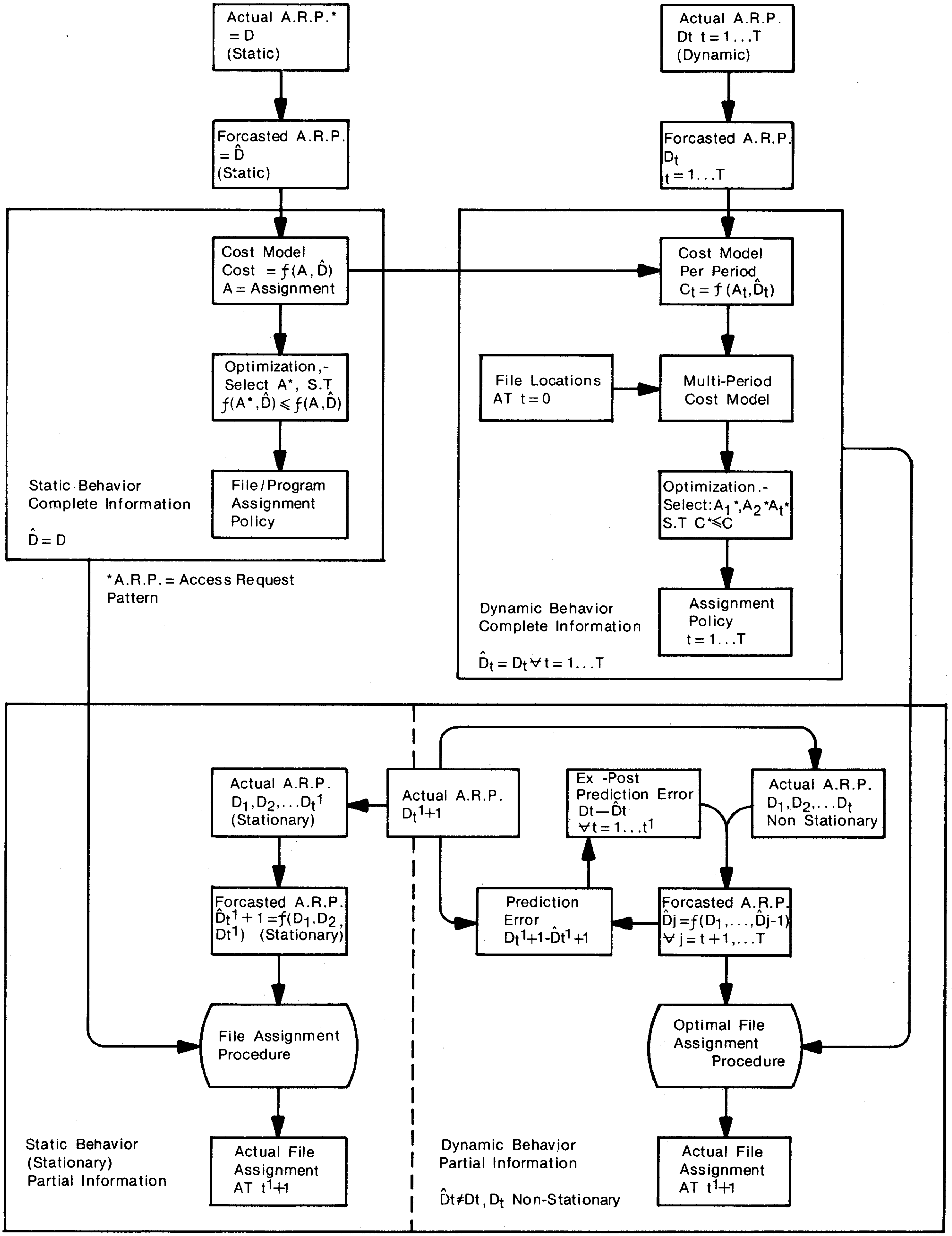

Actual A.R.P.* = D (Static)
Forcasted A.R.P. = D̂ (Static)
Cost Model Cost = ƒ(A, D̂) A = Assignment
Optimization,- Select A*, S.T ƒ(A*,D̂) ≤ ƒ(A,D̂)
File/Program Assignment Policy
Static Behavior Complete Information D̂ = D
*A.R.P. = Access Request Pattern
Actual A.R.P. Dt t = 1...T (Dynamic)
Forcasted A.R.P. Dt t = 1...T
Cost Model Per Period Ct = ƒ (At,D̂t)
File Locations AT t = 0
Multi-Period Cost Model
Optimization.- Select: A1*,A2*At* S.T C*≤C
Assignment Policy t = 1...T
Dynamic Behavior Complete Information D̂t = Dt ∀ t = 1...T
Actual A.R.P. D1,D2,...Dt1 (Stationary)
Actual A.R.P. Dt1+1
Ex -Post Prediction Error Dt—D̂t ∀t = 1...t1
Actual A.R.P. D1,D2,...Dt Non Stationary
Forcasted A.R.P. D̂t1 + 1 = ƒ(D1,D2, Dt1) (Stationary)
Prediction Error Dt1+1-D̂t1+1
Forcasted A.R.P. D̂j = ƒ(D1,...,D̂j-1) ∀ j = t + 1,...T
File Assignment Procedure
Optimal File Assignment Procedure
Static Behavior (Stationary) Partial Information
Actual File Assignment AT t1+1
Dynamic Behavior Partial Information D̂t≠Dt, Dt Non-Stationary
Actual File Assignment AT t1+1

Figure 2—Gross structure of model development

the first moments of the distributions are subject to changes over time.

Recall that the cost associated with a given combination of file assignments over T periods $= G(\mathbf{K}_T)$ is

$$G(\mathbf{K}_T) = \sum_{t=1}^{T} [\sum_{Pq} \sum_{i} \lambda_{ipt} \mathrm{Min}_{j \in J_{Pq}} (C_{ij} + \alpha^* \mathrm{Min}_{k(t) \in K_t} C_{jk(t)})$$

$$+ \sum_{Pu} \sum_{i} \lambda_{ipt}' \mathrm{Min}_{j \in J_{Pu}} (c_{ij}' + \beta^* \sum_{k(t) \in K_t} C_{jk(t)}')$$

$$+ \sum_{k(t) \in K_t} (\sigma_{kt} + \mathrm{Lm}^* \mathrm{Min}_{k(t-1) \in K_{t-1}} C_{k(t-1),K(t)})]$$

Since λ_{ipt} and λ_{ipt}' are random variables, the cost of a given T-assignments (over T periods) is in itself a random variable with yet unknown expectation. Under these circumstances, our objective should be modified and a search should be conducted to find that combination of file assignments that will yield the minimum expected cost.

i.e., Find $\mathbf{K}_T^*$ such that $E[G(\mathbf{K}_T^*)] = \mathrm{Min}_{\mathbf{K}_T} \{E[G(\mathbf{K}_T)]\}$

It has been proven by Levin[5] that the problem of optimal file assignment with stochastic demand patterns reduces to one of estimating the first moment of the access rates' distribution. A statistical procedure for the estimation of these patterns is developed there and incorporated in the file location model.

SOME RECENT RESULTS

Figure 2 describes the various stages of our own research in this area. In Morgan and Levin[6] we have presented results for the file assignment problem under static access request behavior and complete information. A search procedure for finding the optimal file assignment (with respect to minimizing operating costs) is presented there. This procedure is computationally efficient, and we are now experimenting with it on various problems.

Building on this work, we have developed a dynamic model for the multi period case Levin and Morgan.[7] In this model, the access time requests are assumed to be known for the next T periods. However, the assumption that the access request patterns are static over time was relaxed and a dynamic model which considers transition costs was suggested. A dynamic programming solution approach is discussed there with respect to computational efficiency and storage requirements. Figure 2 shows the connection between the static and dynamic models.

Finally, we have been able recently to work on an adaptive model for organizing distributed data bases in computer networks. A forecasting method which attempts to predict the best policy on the basis of where requests are expected to arise is developed. As new information becomes available, the forecast is revised and the assignment policy adapts itself to the new information. Such an adaptive strategy has proven quite efficient in optimizing access time in a memory hierarchy on a single machine (Morgan and Kennedy[8]), and we are hopeful that similar gains can be shown in the network environment.

POSSIBLE APPLICATION AND FURTHER RESEARCH

This research can be applied by both a potential user of an existing computer network and a network designer. From the user's viewpoint, these models can be applied directly for different classes of decisions. A potential user will probably be faced with a set of network specifications, such as prices for storage and communication and expected delays on the communication links. His decision problems range from where to store his files, given a set of nodes which can execute his programs, to the major decision of whether to join the network at all.

From the network designer's viewpoint, these models can be used as the building blocks in an integrated design approach. One can envision an interactive design system in which the network designer sits at a terminal and enters different network specifications, i.e., pricing factors and expected delays. The file assignment models of this research are then invoked to provide him with the revenue figure for the forecasted demand. In addition, the routing discipline and the file assignments determined by these models are transferred as capacity constraints to minimum cost models of channel and storage capacity. The resulting cost figure is communicated to the network designer, with a list of the shadow prices of the relevant constraints. In response, the designer can utilize his judgment to readjust the pricing scheme (or other specifications) in order to relax some of the more costly constraints for another iteration of the models.

Further research in this direction is very promising and the authors, themselves, are currently trying to incorporate these models in an interactive design process.

REFERENCES

1. Chu, W. W., "Optimal File Allocation in a Multiple Computer System," *IEEE Trans. on Computers*, October 1969, pp. 885-889.
2. Casey, R. G., "Allocation of copies of a file in an information network," *SJCC 1972*, AFIPS Press, Vol. 40, 1972.
3. Whitney, V. K. M., *A Study of Optimal File Assignment and Communication Network Configuration*, Ph.D. Dissertation, University of Michigan, 1970.
4. Peebles, R. W., *Design Considerations for a Distributed Data Access System*, Ph.D. Dissertation, Moore School of Electrical Engineering, University of Pennsylvania, 1973.
5. Levin, K. D., *Organizing Distributed Data Bases in Computer Networks*, Technical Report No. 74-09-01, Dept. of Decision Sciences, The Wharton School, University of Pennsylvania, (Ph.D. Dissertation).
6. Morgan, H. L. and K. D. Levin, *Optimal Program and Data Locations in Computer Networks*, Technical Report No. 74-10-01, Dept. of Decision Sciences, The Wharton School, University of Pennsylvania. Presented at the TIMS XX1 international meeting, San Juan, October 1974.
7. Levin, K. D. and H. L. Morgan, *Dynamic File Assignment in Computer Networks under Varying Access Request Patterns*, An invited paper, ORSA/TIMS Joint National Meeting, Chicago, April 1975.
8. Morgan, H. L. and S. R. Kennedy, *An Adaptive File System*, Information Science Dept., Technical Report No. 4, CalTech, pp. 286-80, 1972.

Structured organization of clinical data bases

by GIO WIEDERHOLD and JAMES F. FRIES
Stanford University
Stanford, California

and

STEPHEN WEYL
Stanford Research Institute
Menlo Park, California

INTRODUCTION

The health care delivery system is under strong pressures from several sides. Many of these pressures derive from the demand for a more comprehensive range of health services and from the increased complexity of disease and treatment patterns. Since medical science has provided tools to manage many of the once common diseases, it now has to cope with problems of less well understood origin and course.[1] The practicing physician is faced with an information explosion of major dimensions and a gap between scientific knowledge in a basic form and its practical application at the bedside.[2]

The medical system has responded in part by restructuring itself to provide increasing specialization, allowing the physician to develop a deeper understanding of a narrower portion of medicine. This has resulted in the development of discrete specialized units in the health care system. Specialized diagnostic centers and clinics are examples of this development in outpatient care. These individual units are highly sophisticated, justified by their ability to deliver superior care within the narrower area, but with unique sets of requirements and functions. Within these units work physicians, physician aides, nurse practitioners, specialized therapists, and medical clerical personnel, each of whom interacts with certain portions of the data in the medical record.[3] As the government's share of health support becomes more significant, the increased record keeping required for fiducial accountability adds further burdens to the systems used to collect and maintain data on the care given to patients.

Information needs of the several potential recipients of data from the medical record will vary greatly depending upon the nature of the problem and the purpose of the consultation. Most medical data base systems have concentrated upon the individual patient record.[4] Research oriented systems, on the other hand, have collected smaller, predefined sets of patient data, which then are accessed selectively by element.[5] A related approach involves the collection of information from multiple individual records to collect past experience to facilitate the clinical decision.[6] There are large numbers of systems which mainly collect charges and prepare bills.

Functions of the medical record

Hospital medical record rooms contain a collection of individual patient record folders, which are retrieved for patient care when a patient is scheduled in a clinic or admitted to a bed in the hospital. It typically takes more than one hour to deliver the record for a given patient and months to collect data to study a disease or treatment modality. The inaccessibility of the information in the record room and in the medical record itself prevents utilization of the stored data in response to many other needs.

The medical record room is, however, an important clinical resource. The accumulated clinical experience of many patients over many years contains data potentially relevant to nearly every clinical problem. Epidemiological studies are an important component of the understanding of disease. The majority of published clinical studies are tabulated from such data, despite the haphazard collection of information, the lack of standardization of terminology in the record and a variety of inherent biases in the approach. The volume of records which had to be collected to provide statistically legitimate information to establish the linkage between smoking and lung cancer indicates the massive efforts which will be required for more obscure diseases.

The medical record also fulfills a number of legal requirements. Restrictions imposed on the record due to this function are often cited as limitations to innovation in medical record management. An example is the need to keep signed medical care orders which inhibits merging of the request with the resulting report.[7]

In addition to access to the data of previous diseases, the ideal medical record also preserves a record of the clinical reasoning used to manage the medical problems presented. Since the medical record is intended to be complete, it also provides the full context for the recorded data.

Increasingly, the traditional reliance upon clinical diagnosis is being recognized as inadequate. New classifications of patients with regard to the specific problems they manifest and the specific symptom complexes they demonstrate have greater utility. Medical reference material and abstracts of medical records are also keyed to diagnostic codes and cannot provide data organized to the unique requirements of the individual patient.[8]

Due to the high cost of health care, questions are being asked about the effectiveness of the various components of the care system. Ultimately, each specific element in the health care delivery process must be analyzed for its effect upon the outcome of care. A clinical data base can record utilization of various health care resources and provide measures for rational allocation. The diversity and drama of the medical environment, together with political and social pressures, can easily distort rational priorities if good data on cost effectiveness are not available.[9]

Difficulties with automated clinical data bases

It seems surprising that few comprehensive data base systems in medicine have been implemented. A number of barriers have limited the development of systems able to support the functions previously enumerated.

An important problem exists in the technology itself. Whereas we have developed procedural programming languages to the extent that simple computer applications can be efficiently and reliably developed, in practice a single individual has to be able to understand the entire scope of the system.[10] Small to moderate computer applications can be efficiently developed and achieve adequate reliability; for larger systems formal techniques of modularization are required.[11] Data base systems have not yet been definitely separated into appropriate levels of abstraction, as have areas as pattern recognition applications and design automation systems.[12]

Exact specifications of the uses to be made of stored medical information have not been available. The programmer has seldom been able to determine what the physician wants, and the physician himself has been unaware of what he needs. Clinical data base systems have often been archival in nature and have not been effectively used for data retrieval purposes.

Extraction, verification, decoding, and transformation of available data has to be supported so that statistical procedures can operate on subsets of stored data. The unavailability of highly interactive systems with large files has prevented the development of approaches in statistical data handling which would exploit more fully dynamic access to data.[13] The potential of graphics to bridge the gulf between the results of data analysis and their clinical applications has also not yet been realized adequately, again partially due to the high cost of the few model systems.[14] These limitations have discouraged the clinician from participating fully in development of clinical data systems. Without an adequate cadre of medical users, medical data base systems have generally failed to have a significant clinical impact, which has in turn reduced their cost-effectiveness.[15]

There is a high cost in the collection of data, and additionally an equally high cost in the operation of a system which verifies input, provides feedback in a useful form and encourages correction of data. Without assurance of fruitful utilization of information, it has been hard to justify the systematic collection of large amounts of medical data.[16] Systems storing small amounts of data have attacked specific areas of interest to a clinician or researcher, but have generally not been clinically useful. Attempts to store large amounts of data have not demonstrated cost-effectiveness. Large systems have interrupted the clinical habits of the physician without providing him with usable information in return. Information in many medical data bases is limited to few variables and often financially oriented or restricted to small sample populations.

DEVELOPMENTS AT STANFORD

Availability of a highly interactive time-shared system with the capability to process large files with minimal difficulty has allowed a number of medical researchers to implement small data bases oriented toward their specific needs.[17,18] After a number of years of availability of the time-shared system, a pattern of data base usage began to emerge. Clinicians began to develop individualized, highly structured, and clinically satisfactory medical record systems prior to approaching the problems of efficient computer data storage and generalized retrieval capabilities. Systems developed within specialty areas with immediate relevance and satisfaction to the medical users.[19] Common patterns of information handling desired by the physicians from the computer databank began to be ascertained. Physicians in specialties not possessing these systems wanted to have access to the benefits while maintaining independence in defining their specific data collection and data analysis requirements.

The schema based approach

In order to achieve commonality of data base use and to provide a basis for controlled future development, a data base support system based upon the concept of a "schema" was developed and put into operation.[20] The design recognizes four levels of abstraction:

- application programs which refer to data by name.
- service programs which use the schema to interact with the data base.
- a file system, summarized later.
- a storage management level which allocates and moves blocks.

A Time Oriented Data Base (TOD) contains clinical information organized chronologically and generally collected by the physician in the form of a time oriented flow sheet of clinical data. Two levels of hierarchy are recognized within a clinical data base:

- the patients
- the visits of the patients to the clinic

A schema is a structure which defines the specific content of the individual databank. The two levels used by TOD are described in an identical manner.[21]

An entry in the schema for a data base element specifies:

- A descriptive name for the element
- A short, formal name
- The units of measurement for the element
- The element's data type, oriented to user requirements, rather than to available hardware operations
- Range limits for numeric data
- Coding requirements for non-numeric values
- The retrieval files in which the data element is recorded to meet accessing requirements

In addition, the data base manipulation programs collect into audit record associated with the schema various utilization data and integrity indicators. Each complete schema provides a formal and machine-processable description of each data base. This description is used by the processing programs to identify, select, and analyze patient and visit records as well as specific data elements within these records. All data bases which are described using the schema and which follow certain structural rules can use these processing programs without any user-specific changes. There is no generalized data base management system to provide these services. There is instead a library of programs which is shared by the various databank users to the extent that they provide useful services to them.

The authority over contents, the responsibility for updating, and maintenance of quality remain in the hands of the "owner" of each of the data bases. Programs developed by one user can be shared by all, or copied and modified to provide information and formats suitable for one specific clinic. The databank is selected by the user rather than being imposed upon him and he is not threatened by arbitrary decisions made by others. In practice, a rather remarkable diversity has been evident. TOD structured databanks have been implemented for more than a dozen clinics and research groups within the Stanford Medical Center. Unanticipated applications have been in the office of the dean of the medical school to keep records on incoming students and faculty appointments as well as a system to control hospital engineering projects.

Since all medical data bases serve subgroups of the same regional patient population, another hierarchical level is available which addresses the problem of communication and centralization. A central management function has access to the individual schemas and to their utilization records. There are a number of programs here which extract and aggregate this information. This also allows control and advice-giving regarding naming and coding conventions. A formal name for each data domain provides the basis for inter-data base linkages and hence is subject to joint agreement. All coding and decoding of data entries is carried out within the TOD system, as specified by the schema, so that these procedures are also sharable. This feature aids in the maintenance of internal and external consistency between otherwise separate systems. The utilization data can provide guidance to management when system extensions or changes are being considered.

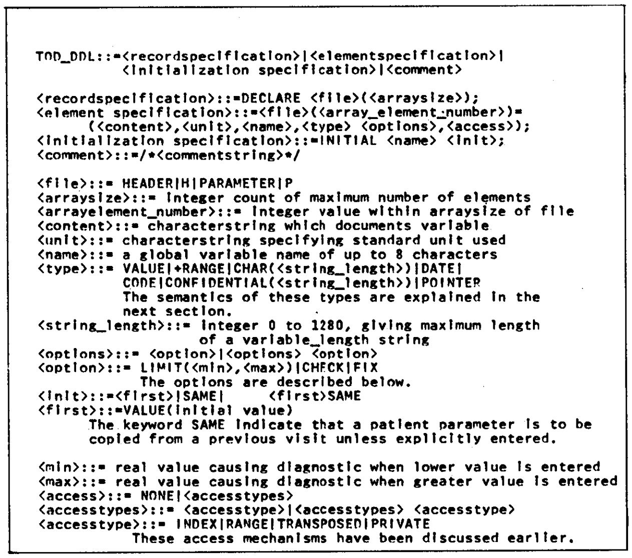
Structure of the TOD files

The files used by a TOD data base can be categorized into three areas:

- The Schema Files, describing the detailed structure.
- The Data Files, containing the actual data values.
- The Accessing Files, which provide rapid access to data according to various retrieval criteria. All data in these files are redundant relative to the data files themselves.

The schema resides in two files. The first is the source file, containing TOD Data Definition Language (TOD_DDL) statements. It provides the documentation, and contains lines of text, normally in record element order. The second, object, file is produced by the TOD_DDL translator, and is oriented toward efficient driving of the interpretive TOD data entry and extraction routines. The record content specification here is arranged in transposed form; for instance, the type codes for all elements of the record for a file appear in a single object schema record. Only the translated file is used during TOD processing. The object form can be fully detranslated to the source schema; even comments are preserved in designated object schema records.

All data reside on two files: the HEADER file, which contains non-chronological patient information, and the PARAMETER file containing visit information. The HEADER file refers to the PARAMETER file using symbolic numeric keys, visit numbers, which are used to locate the records via the index structure of the parameter file. This provides the linkage without placing restrictions on the organization or placement of the parameters. The first and last visit key is specified in the header, and the other visits in the parameter file are chained both forward and reverse, again using symbolic keys, which allows efficient retrieval both in chronological order, as well as for a series of recent visits preceding the last visit. Both header and parameter records have in their identification field a unique identifying value for each patient to allow recovery if chains should be damaged. This simple structure avoids the need for complex hierarchies and repeating groups.[22,23] Other criteria for normality are met since the data elements within a record are due to independent observations. Research on the data could of course discover functional dependencies!



```
TOD_DDL::=<recordspecification>|<elementspecification>|
          <initialization specification>|<comment>

<recordspecification>::=DECLARE <file>(<arraysize>);
<element specification>::=<file>(<array_element_number>)=
      ((<content>,<unit>,<name>,<type> <options>,<access>);
<initialization specification>::=INITIAL <name> <init>;
<comment>::=/*<commentstring>*/

<file>::= HEADER|H|PARAMETER|P
<arraysize>::= integer count of maximum number of elements
<arrayelement_number>::= integer value within arraysize of file
<content>::= characterstring which documents variable
<unit>::= characterstring specifying standard unit used
<name>::= a global variable name of up to 8 characters
<type>::= VALUE|+RANGE|CHAR(<string_length>)|DATE|
          CODE|CONFIDENTIAL(<string_length>)|POINTER
          The semantics of these types are explained in the
          next section.
<string_length>::= integer 0 to 1280, giving maximum length
                   of a variable_length string
<options>::= <option>|<options> <option>
<option>::= LIMIT(<min>,<max>)|CHECK|FIX
            The options are described below.
<init>::=<first>|SAME|     <first>SAME
<first>::=VALUE(initial value)
      The keyword SAME indicate that a patient parameter is to be
      copied from a previous visit unless explicitly entered.

<min>::= real value causing diagnostic when lower value is entered
<max>::= real value causing diagnostic when greater value is entered
<access>::= NONE|<accesstypes>
<accesstypes>::= <accesstype>|<accesstypes> <accesstype>
<accesstype>::= INDEX|RANGE|TRANSPOSED|PRIVATE
           These access mechanisms have been discussed earlier.
```

Figure 1—Syntax for the TOD data description language

An optional data file, the OVERFLOW file, is only used if non-numeric parameter entries are kept without encoding. Then a reference key is inserted in the record position and the string itself is stored in the overflow file. Since there is no satisfactory analysis possible for data which do not lend themselves to encoding, there seems to be no reason to keep values of these parameters in the prime data file. String data are directly stored in the HEADER file, since such information is useful for searching and classification.

Accessing files are created on request. These contain data elements, which were indicated in the schema to be important search keys, arranged to optimize retrieval, and are used, when available, by the data retrieval programs. These files are typically generated overnight. They are identified with the date of their creation. This allows reporting of the date of validity in reports which use them. Both initial and final generation time are written into identification records of these files to provide verification of update process completion.

> An INDEX file contains all values of the specified element, in ascending order, with the symbolic keys to the records. Any PARAMETER or HEADER element may be indexed.
>
> RANGE files contain the lowest and highest value of the specified PARAMETER element of one patient, and provide the key of the patient HEADER. This requires less space than the INDEX and is especially suitable for case selection.
>
> The TRANSPOSED file is organized by element. A record consists of all entries for the specified element. This facilitates efficient and rapid retrieval according to complex criteria based on a few parameters. No keys are provided since the corresponding PARAMETER record can be found positionally. The transposed files have become the principal means for selection and data analysis since statistical procedures can operate on transposed data without requiring a reference to the primary data file.
>
> If desired, additional PRIVATE access files can be constructed, using the TOD scheduling and verification facilities.

Other accessing files are generated to support the subsetting capabilities of the analysis programs*. Subsets that are created are catalogued and identified with the information from the request which generated them and with specifications from the schema. These facilities are described in the TOD User's Manual.[24] This documentation also describes the supporting data entry, analysis, report generating, and graphics output procedures.

The data description language

The data description language is quite simple, so that after initial definition, maintenance of the schema can be carried out by the owner of the data base.[25]

The syntax for the language is sketched in Figure 1.

The data types used within TOD are oriented toward the medical user:

> VALUE data items can meaningfully assume rational numbers representing continuous values. These numbers are of the standard IBM 360 precision of not quite 7 digits.
>
> +RANGE data items can assume numeric values between 0 and 4. This implements a common notation used to indicate severity of problems or symptoms.
>
> DISCRETE data items can only meaningfully assume discrete numeric values. Statistical procedures using these data are restricted to classification and similar operations.
>
> CHAR(string_length) specifies that a data item has as its value a string of variable length less than or equal to string_length. This data type is natural for HEADER elements, and not recommended for PARAMETER elements.
>
> DATE data items store dates in a numeric form. Dates are displayed in a standard format, DDMONYY. DD are two digits specifying the day, MON is first three letters of the month, and YY are the last two digits of the year. There are no spaces. On input this and unambiguously similar forms are automatically encoded into a Julian date for storage and numerical manipulation. The use of an alphabetic representation for the months is intended to reduce confusion which can easily occur due to the diversity of national backgrounds found in the medical environment.

* The subsetting capabilities have recently been greatly extended by Alison Harlow of the Department of Immunology, and will be the subject of a separate paper.

CODE data items are stored as numeric values related by encoding-decoding procedures to a more meaningful representation. For example, the item "sex" might be coded as 0, 1 for Female, Male. This is achieved by invocation of an ENCODE or DECODE procedure at data-entry or output time. These procedures receive as arguments the array element_number as well as the value of the input string or array element.

CONFIDENTIAL data items are encoded and decoded by procedures providing keywork-protected scrambling. The procedure may be a general Vignere ciphering procedure provided by the system. The keystrings, of course, are private and submitted by the user at the first instance of a scrambling request.

POINTER data items store pointers to information contained elsewhere. What gets pointed to by a POINTER item is entirely up to the individual user. As an example, a HEADER element of type POINTER might point to the first of a group of text-file lines comprising a reference letter for each patient. This facility is only intended as an escape hatch to implement non-standard features, and is yet unused.

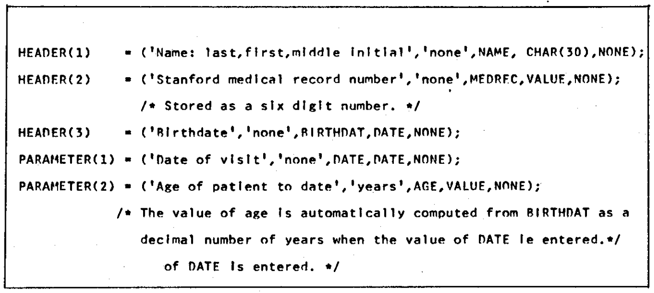
Three HEADER and two PARAMETER element assignments must be included in every databank (Figure 2) if a user wishes to take advantage of public data entry and retrieval programs. The assignments are written below in TOD_DDL with explanatory comments.

A trivial example of a schema for a databank is given in Figure 3.

The options, LIMIT, CHECK and FIX specify data checking procedures. LIMIT and CHECK are invoked at data-entry time. The option CHECK causes invocation of a procedure in a manner similar to the CODE data type specification. The CHECK procedure then can apply arbitrarily complex verification procedures. It can access any data value in the data base to accomplish its task.

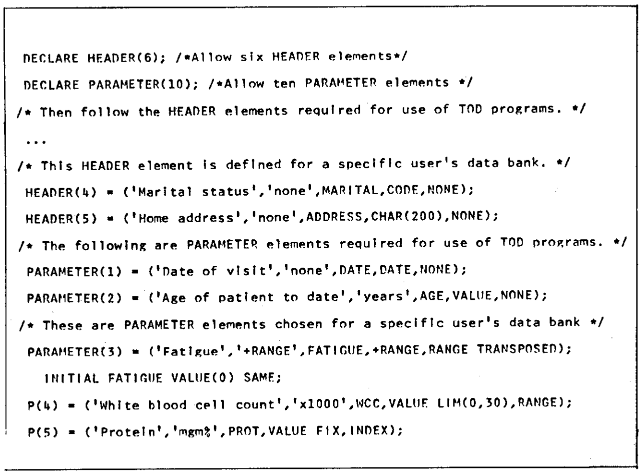
The CODE procedures themselves, when they are invoked, will also provide data input filtering capability. Both are structured so that they can reprompt the user for correct data values.

The FIX specification invokes checking procedure in a deferred operation. Since extensive on-line checking can make a system unresponsive and costly, there can be a significant benefit if complex checking procedures are executed at night or other low utilization times. An error report will be produced, and when correct values have been determined, they can be entered via the normal update mechanism.



```
HEADER(1)     = ('Name: last,first,middle initial','none',NAME, CHAR(30),NONE);
HEADER(2)     = ('Stanford medical record number','none',MEDREC,VALUE,NONE);
                /* Stored as a six digit number. */
HEADER(3)     = ('Birthdate','none',BIRTHDAT,DATE,NONE);
PARAMETER(1) = ('Date of visit','none',DATE,DATE,NONE);
PARAMETER(2) = ('Age of patient to date','years',AGE,VALUE,NONE);
              /* The value of age is automatically computed from BIRTHDAT as a
                decimal number of years when the value of DATE le entered.*/
                  of DATE is entered. */
```

Figure 2—Required header and parameter items



```
 DECLARE HEADER(6); /*Allow six HEADER elements*/
 DECLARE PARAMETER(10); /*Allow ten PARAMETER elements */
/* Then follow the HEADER elements required for use of TOD programs. */
 ...
/* This HEADER element is defined for a specific user's data bank. */
 HEADER(4) = ('Marital status','none',MARITAL,CODE,NONE);
 HEADER(5) = ('Home address','none',ADDRESS,CHAR(200),NONE);
/* The following are PARAMETER elements required for use of TOD programs. */
 PARAMETER(1) = ('Date of visit','none',DATE,DATE,NONE);
 PARAMETER(2) = ('Age of patient to date','years',AGE,VALUE,NONE);
/* These are PARAMETER elements chosen for a specific user's data bank */
 PARAMETER(3) = ('Fatigue','+RANGE',FATIGUE,+RANGE,RANGE TRANSPOSED);
   INITIAL FATIGUE VALUE(0) SAME;
 P(4) = ('White blood cell count','x1000',WCC,VALUE LIM(0,30),RANGE);
 P(5) = ('Protein','mgm%',PROT,VALUE FIX,INDEX);
```

Figure 3—Example of a schema

File service

The implementation of TOD would not have been feasible without the availability of the services of a powerful file system.[26] The essential features of the ACME file system will only be summarized.

The file system provides isolation between users. Files which have not been further protected, however, can be read by other legal users of the system. The general public library is actually handled as one additional user. Nearly absolute reliability has been achieved. The use of programmed redundancy, internally suspicious code, system initiated back-up procedures, and other techniques has resulted in a file system, which through many years of normal operations with the typical rate of equipment changes and failures, is trusted by the medical staff.

The file structure supports sequential access, direct access and indexes all within one physical file design. Sequentially written records are given ascending integers as keys, and the index allows direct retrieval of compactly stored variable length records, assigned dynamically to not necessarily contiguous blocks. The two level index supports only numeric keys. The key for the patient file is a patient number and the key for the visit file is a sequential visit number. The files, while not physically sequential, appear hence as indexed-sequential files.

The availability of compression makes it possible to store the inherently sparse medical records efficiently, while the user works with array oriented concepts which are convenient to manipulate.[27,28] The compression of numeric data uses a four-valued coding system:

00	value is zero
01	value is undefined
10	non zero value
11	repeat of previous value

The "undefined" data type (coded within the processor as negative zero) is recognized throughout the ACME system, and is specifically important in statistical analysis. The compression, even with the additional bits required for new non-zero values, reduces TOD data files by a factor of five. The access files are also compressed. They contain many repeated element values.

The fact that compression is employed causes changes in the sizes of the records when the records are updated. An implementation of the compression algorithm for dynamically changing files is only possible because the design óf the file system permits the rewriting of records that have changed in size. Few file systems permit such an action. Occasionally data have to be moved between blocks or new blocks have to be logically inserted to enable record expansion.

The use of these file system facilities permits the presentation of the formal structure to the users, without burdening their programming with file efficiency considerations, which are user and environment specific.[29] The levels of hierarchy of the data have also been implemented through symbolic linkages, rather than through the use of pointers. This structure provides more flexibility in data management. We believe that these are important aspects of a methodical approach to the development of successful large information systems.

Operational support

The entire system is written in an interactive subset of PL/1, called PL/ACME, at the Stanford University Medical School.[30] The TOD system was originally implemented using the ACME services on an IBM/360, model 50. The ACME System operated as one partition under IBM's OS system. Stanford has since replaced the model 50 with an IBM 370, model 158 and ACME runs there under VS2, using mainly virtual storage, but it still maintains its own file system. It would seem desirable to use the IBM VSAM file system instead, in order to provide system compatibility, albeit at higher overhead costs. Conflicting demands on the 158 system have caused Stanford to decide to implement ACME-like files and other facilities on an IBM 360, model 67. No IBM file system exists for the 360 which offers adequate facilities, so that again a special file system is required. Uncertainties associated with the hardware changes described above have retarded the ongoing TOD development.

Currently, all TOD users share the same large computer. For applications where a dedicated smaller system is more appropriate, the concept of the schema and the common data names provides the same benefits and may be an important step toward communication between data bases which are both logically and physically distant. Since the intensity of data utilization tends to be highest within individual project areas, a modest level of intersystem communication capability should be adequate to provide data sharing when required. Such alternatives are being investigated.

CONCLUSION

The system described attempts to allow the variety of access and use seen in the medical record environment while providing many sharable resources. The structure imposed: 'patient having visits' seems well accepted. Within this structure we find databanks specifying from 20 to 500 data items which might be collected per patient visit. The usage of the databanks ranges from the monitoring of patients follow-up visits to the prognosis of disease development from the records for patients which presented similar symptoms. This is done without dependence on diagnostic codes.

The demonstrated ability to share programs allows new projects to reach operational status rapidly after file definition and user-specific programs have been completed. Most databanks do not have a full-time programmer associated with them. The owners have been willing to pay for the computer services at approximately commercial rates. There is only one user outside of Stanford University and promotion of such use is not within the mandate of the institution.

The adequacy of the approach is best demonstrated by the extent with which such systems are accepted and adapted to cope with the changing medical environment. Incompatibility and weaknesses in the technical support still inhibit direct transfer of interactive data base applications. Documentation of effective computer applications will provide directions for further development of appropriate systems.[31]

ACKNOWLEDGMENTS

Many people have contributed to this effort. We wish to thank especially Frank Germano, who expended much energy in bringing the system to a routine operational level. All of the users have contributed more than they intended. This description of the system was improved greatly by the thorough work of the NCC reviewers. Applications of the system have been reported elsewhere by Fries and Weyl among others.[32,33,34]

Current developments of the system are in the hands of Ned Russell of the Stanford Center for Information Processing and Jim Standish and Alison Harlow of the Division of Immunology, Stanford University Medical School. Initial resources for the implementation of TOD came from the Biotechnology Resources Branch of NIH

(RR 00311-4) which established the entire ACME service. Present support is afforded by an award of the American Society of Computer Medicine.

REFERENCES

1. Starmer, C., R. A. Rosati, J. F. McNeer, "Data Bank Use in Management of Chronic Disease," Editorial, *Computers and Biomedical Research*, Vol. 7, No. 2, April 1974.
2. Feinstein, A. R., *Clinical Judgement*, Wilkins and Wilkins, Baltimore, 1967.
3. Somers, Anne R., *Health Care in Transition, Direction for the Future*, Hospital Research and Educational Trust, Chicago, 1971.
4. Davis, Lou, "Prototype for Future Computer Medical Records," *Computers and Biomedical Research*, Vol. 3, No. 5, Oct. 1970.
5. Horton, Caroline L., J. Edward Okies, Bruno J. Messmer, Grady L. Hallman and Denton A. Cooley, "Computer Application in Clinical Medicine, CARE: A Practical System for Processing Data," *Computers and Biomedical Research*, Vol. 6, No. 3, June 1973, pp. 286 to 298.
6. Fries, James F., "Experience Counting in Sequential Computer Diagnosis," *Archives of Int. Med.*, 1970, pp. 647 to 651.
7. Freed, Ray N., *Law and Contemporary Problems*, Autumn 1967, Duke University School of Law, Durham, N.C.
8. Weed, Lawrence L., *Medical Records, Medical Education and Patient Care*, Case Western Reserve University press, 1969.
9. Culliton, Barbara J., "Cancer, Heart Disease, and Everything Else," *Science*, Vol. 181, Aug. 1973, pp. 822 to 830.
10. Baker, F. T., "Chief Programmer Team Management of Production Programming," *IBM Systems Journal*, Vol. 11, No. 1, 1972.
11. Parnas, D. L., "On the Criteria to be Used in Decomposing Systems into Modules," *Comm. of the ACM*, Vol. 15, No. 12, Dec. 1972.
12. Wiederhold, Gio, *A Proposal for the Structuring of the Data Base Problem*, April 1973, unpublished, available from the author.
13. Wiederhold, Gio, "New Environments for Statistics," Allied Statistical Meeting, Detroit 1970, abstract Stanford Univ. ACME Note ASD, July 1970.
14. Dixon, Wilfrid J., "Use of Displays with Packaged Statistical Programs," *Proc. of the 1967 FJCC*, AFIPS Vol. 31, pp. 481 to 489.
15. Barnett, G. Octo, "The Use of Computers in Clinical Data Management: The Ten Commandments," *AMS Conference*, Las Vegas, Nevada, Feb. 16, 1971.
16. Baker, M. G. J., et al., *Data Base Management Systems, Their Use in Hospital Data Processing*, Her Majesty's Stationery Office, London 1972.
17. Sanders, W. J., Gio Wiederhold, et al., "An Advanced Computer System for Medical Research," *Proc. of the 1967 FJCC*, AFIPS Vol. 31, pages 497 to 508.
18. Frey, Regina, Serge Girardi and Gio Wiederhold, "A Filing System for Medical Research," *Int'l Journ. of Biomedical Computing*, Vol. 2, No. 1, 1971.
19. Fries, James F., "Time-Oriented Patient Records and a Computer Data Bank," *Journal of the Am. Medical Assn.*, Vol. 222, No. 12, Dec. 1972.
20. CODASYL Data Base Task Group, April 1971, ACM, New York.
21. Weyl, Stephen, *Definition of the PL/ACME Time-Oriented Data Bank Protocol*, Stanford University, ACME Note TODD-1, Oct. 1972.
22. Codd, E. F., "A Relational Model for Large Shared Data Banks," *Comm. of the ACM*, Vol. 13, No. 6, June 1970.
23. Codd, E. F., "Further Normalization of the Data Base Relational Model," in Randall Rustin *Data Base Systems*, Prentice Hall 1971.
24. Germano, Frank, Stephen Weyl, Voy Wiederhold, et al., *Stanford University TOD (Time-Oriented Databank) Users Manual*, April 1973, Aug. 1973 (2nd ed.).
25. Wiederhold, Voy, *How to Write a Schema for a Time Oriented Medical Record Data Bank (TOD)*, Stanford University, ACME Note TDUA-2, July 1973.
26. Miller, Gerald F., *The ACME File System*, Stanford University, ACME Note FY-1, Feb. 1969.
27. Wiederhold, Gio, *Proposal to Aid in More Efficient Usage of Disk*, Stanford University ACME Note PEFF-1, Dec. 1971.
28. Granieri, Charles, *Data File Compression, Implementation Notes*, Stanford University ACME Note WCOMPS-2, April 1973.
29. Johnson, Martin E. and Ruth E. Dayhoff, *MUMPS Primer*, National Bureau of Standards, Sept. 1974.
30. Breitbard, Gary Y. and Gio Wiederhold, "The ACME Compiler," *Proc. of the 1968 IFIP Conference*, North Holland 1969, pp. 358 to 365.
31. Wiederhold, Gio, *Data Base Structure and Schemas*, in preparation, partially available as class notes for MIS290, USCF.
32. Fries, James F., "Alternatives in Medical Records," *Medical Care*, 1974.
33. Fries, James F., Stephen Weyl, and Hal Holman, "Estimating Prognosis in Systemic Lupus Erythematosus," *Am. Journ. Med.*, Vol. 57, pp. 561 to 565, 1974.
34. Weyl, Stephen, James Fries, Gio Wiederhold, and Frank Germano, "A Modular, Self-Describing Clinical Databank System," to be published in *Computers and Biomedical Research*, April 1975.

Integrated data bases for municipal decision-making

by PATRICK E. MANTEY and ERIC D. CARLSON
IBM Research Laboratory
San Jose, California

APPLICATIONS OF A MUNICIPAL DATA BASE

Municipal governments are, in essence, created to deliver services to a geographical area. There is an unusual variety, in comparison to private industry, of services offered. Local government is often structured (or fractured) along functional lines into special districts, as well as by geography. Such structuring has precluded concentration of power, but it has increased the complexity of planning, resource allocation, or management.

Many of the problems in municipal government require decisions which are not routine. Rather, they require the professional insight and judgment of human decision makers who consider the specific conditions of each problem. Ideally, this insight and judgment would be aided and guided by appropriate information derived from a comprehensive data base. This is the objective of a municipal information system: to facilitate effective analysis and solution of specific problems by supporting human decision makers with data resources and analysis functions in readily usable form. Because a municipality provides services to a geographical area, much of the data relevant to decision-making or problem solving in local agencies will have geographical attributes, and can be given spatial interpretation via maps. A key attribute of a municipal information system is the capability for displaying information in the form of maps. Another requirement for such systems to be effective is that they support ready use by decision makers who know very little about computers. The system must help the decision makers develop their precise objectives or decision criteria for solving a problem. The solution process in such decision-making requires exploratory analysis, selection of relevant data, and meaningful data presentation in an interactive environment. A system which provides such capabilities, called GADS (Geo-data Analysis and Display System) has been developed and evaluated in several applications, such as police manpower allocation and analysis of urban development policies.[1-4] The evaluations of GADS indicate that interactive analysis and display systems have a great potential in the operations, management, and planning of municipalities.

As an example, consider a municipality which maintains a computerized property information file (via the tax assessor function). Such a file would have detailed data on each parcel. If this data were accessed via an interactive information system, a decision maker could readily obtain, for example, the address and assessed value of all residences constructed between 1960 and 1962 and having floor area between 1600 and 1800 square feet, on lots with 6800 to 7200 square feet. If the user of the system were a real estate appraiser, this information displayed on a map would provide the appraiser with information in a spatial framework. If the user of the information system were an assessor concerned with determining the neighborhoods which could be considered equivalent for purposes of computer-aided appraisal, additional data would be required. If recent sales records are the basis for calibrating the assessment model, it may be found that the assessor's data alone cannot be used to model variations in selling price of houses fitting the description above. Showing on a map the mean and variance of the selling price of such houses by neighborhood will offer the assessor a visual means for examining the quality of the assessment model. The display may cause the assessor to consider other factors to explain the variations; e.g., crime rate, level of public facilities and services (such as the influence of an adjacent regional park[5]) or the influence of other near-by land uses. For the user to further investigate the influence of these factors, the information system would have to provide access to the data in the appropriate files (e.g., crime, public facilities, land use) and support the relating of data in these files to the neighborhoods of interest. Such systems, which provide access to arbitrary subsets of a data base and which facilitate the development of such subsets at arbitrary levels of aggregation, will be said to offer a comprehensive data base.

CURRENT STATUS OF MUNICIPAL DATA BASES

The importance of a comprehensive data base to support decision-making in municipal government has been widely recognized. There have been several different approaches taken to the development of comprehensive municipal data bases.

One approach, which was popular in the 1960's, was the development of a comprehensive "data bank." This data bank was usually generated as a special collection or census, and often was carried out and funded as part of a comprehensive transportation and land use planning study. Data acquisition consumed a major portion of the resources of these studies and often did not provide any information relating to many municipal services (e.g., public safety). Although accurate data was often gathered in the development of these data banks, their value was short-lived because they were at best a snapshot of the state of a very dynamic system, and no means were provided for updating and extending these data banks.

At about the same time, the computer was becoming a tool in the operations of local governments. The application of computers by municipalities beginning in the 1960's can be characterized as a function-by-function approach, with data processing introduced into tasks involving high-volume routine transactions. The computer is generally utilized in those functions which have previously been computerized in private industry: payroll, accounting, billing, budget status reporting, personnel records, etc. Also, the 1960's saw widespread use of computers in the processes associated with property records, elections, and in operations of law enforcement agencies. Usually these applications were isolated from each other, and no attempts were made to make this information available for use by other municipal functions.

Current approaches related to the development of municipal data bases are characterized by the USAC projects,[6] particularly those of Charlotte, North Carolina, and Wichita Falls, Texas. These cities were funded by the Federal USAC project to build Integrated Municipal Information Systems, (IMIS). The USAC efforts involve city governments, and were the consequence of studies such as the IBM/New Haven project[7] and the USC/Burbank project.[8] These earlier groups sought to develop a methodology, via a "systems approach", for the application of computers by municipalities[6] but did not result in implementation of integrated municipal information systems. The USAC approach wisely focused on operational sources to provide the current data required for municipal decision-making. In the implementations, which are still in progress, the cities have concentrated on building up operational uses of computers, and on implementing these applications on a central computer under an integrated data-base management system. The value of computers to these operational functions has been confirmed[9] but the applications of IMIS in the areas of management decision-making are still to be demonstrated.

One of the difficulties that must be overcome in providing a comprehensive municipal data base (even in cities with a fully integrated and operational IMIS constructed according to the USAC philosophy), is that complete integration, where all municipal functions use the same computer and data base management system, is not a likely prospect with current local governmental structure and with the limited resources of local governments. For example, the data pertinent to decision-making in a city may be gathered by another agency, such as the tax assessor, and may reside on different computers, under different data management schemes and in different file formats. In addition, problems of data security, compatibility of files, and high processing costs may make complete integration unrealistic for many municipalities. Special data collections, such as the U.S. Census, and data available from state sources, must also be readily incorporated into a municipal data base. With census data gathered according to blocks, block groups and census tracts, with assessors property data coded according to assessor map, book and page, with public works data in state-plane coordinates, and school data gathered by school attendance area, the building and maintenance of a truly integrated municipal data base presents a formidable task.

APPROACHES PERMITTING DEVELOPMENT OF INTEGRATED FILES

A completely integrated data base would have all data relating to any functions of a municipality residing on the same computer system, under the same data management system and organized and indexed to facilitate correlation. This ideal is not attainable, given present organizational structures and computing capabilities in most municipalities. However, if the computerized files of various municipal functions are "properly structured," it will be possible to achieve the same benefits as if there existed a completely integrated municipal data base. In addition, such an approach will not require re-implementation of current applications, but rather leaves the application data base in the control of those responsible for its primary maintenance and use.

The approach taken is to make use of data, when data files are "properly structured," to develop the same results as if there existed a completely integrated data base without requiring that complete integration take place. This should not be construed as an argument *against* integration. If integration is politically and technically possible, provides required data security, and is economically attractive, it should be implemented. Even with an integrated data base, there will always be decisions which require different groupings of data than those supported by the integrated data base. There will also remain, in practice, data sources which cannot be integrated. So, the problem of providing a comprehensive data base from multiple data sources is unavoidable and is not completely solved by an "integrated" data base.

The "proper structuring" required to make data integration possible can be illustrated by example. If one is interested in information about burglaries, and wishes to relate this information to neighborhood conditions, data sources could include police dispatch files, criminal justice arrest files, census data and tax assessor files. Suppose the police dispatch data is used for burglary incidence, and that such data is available in terms of police beats, e.g., the number of burglaries in each beat for each day. If one wishes to use census data for socio-economic information, and if the census tracts and beats have few common boundaries, no small area information is obtainable relating these data sources. Alternatively, if the police dispatch data is captured by the street address of the call, and if a directory exists for the city which will permit identification of the census tract for each street address, then burglaries and socio-economic data can be related at the census tract level.

"Proper structuring" of data files only has meaning with respect to potential uses of the data (i.e., data files are not an end in themselves). If the objective is to offer data to support decision-making in a wide range of problem areas, then the data files must be as detailed as possible, within the constraints of economics, privacy and security. The detailed data can then make possible the development of the widest variety of data subsets and aggregations, and is more likely to permit development of the required set of integrated data for a particular decision-making context. An additional requirement is the existence of data elements in each file which will facilitate relating the data to that from different files. (In this paper, geographical references will be singled out as data elements serving this function in municipal files. Common references to account numbers, project numbers or personnel identifiers are other examples of data elements permitting the relating of data from different source files.) A set of files will be called "properly structured" if it contains information permitting the relating of data from different source files so that integrated subsets of data at the appropriate level of detail can be developed to support the requirements of problem solvers.

Because municipal government is a service delivery function, mutual references to geography can often be used to relate data from the diverse files available in municipalities. A powerful file in facilitating the relating of data, based on these common geographical references, is a Geographic Base File (GBF). Functionally, the GBF contains data to support the relating of data from other files to geographical location and also the display of this data on a map. The creation of a GBF for a municipality is a key requirement in the development of a comprehensive municipal data base from source files. Several different approaches have been taken.

The simplest GBF is a file sometimes called a Property Location Index (PLI) which contains a list of the valid addresses in the municipality and an x,y coordinate for each. This approach is the one used in Lane County, Oregon, and by the Assessor in Santa Clara County, California. To make this more useful, a list of public place and street intersections and their x,y coordinates is appended. With such a GBF it is then possible to automatically convert addresses (in the police call file for example) to x,y coordinates. If the GBF also contains the police beat, census tract, and municipality for each address, then it is very simple computationally to count the number of calls in each beat. Evaluation of calls by census tract would also permit consideration of socio-economic data with the crime data (of course, police officers could also encode calls by beat and census tract, but this approach is liable to significant errors and seems to be a poor use of police manpower).

The most detailed GBF's contain digitized land parcel boundaries, easement locations, building outlines, utility placements, and even topographic information, along with street address information on all parcels and names of all public lands and buildings. This GBF is at the level of detail of surveyor's data, and is suitable for engineering applications and detailed map building. Ottawa, Canada's National Capital Commission[11] has pioneered in the development of this kind of GBF.

The most common GBF at the present time is the result of work by the U.S. Census Bureau in conjunction with the 1970 Census. Using the Metropolitan Map Series, a massive feature labeling and digitization was performed for 200 major metropolitan areas in the United States. The resulting computerized maps were called the DIME (Dual Independent Map Encoding) files.[12] Each entry (record) in the DIME file represents a line segment (a portion of a street segment, railroad, creek, city limit, etc.). Administrative overlays (e.g., beats, census tracts) can be readily defined in terms of segments of this file. Used in combination with "point-in-polygon" routines, these computerized overlays facilitate development of counts of events in areas of any specific overlay map.

As in the development of any large machine readable file, high startup costs, data errors, and poor standardization have hindered development of GBF's. But the key problem in the development and use of a GBF is editing (corrections and additions). Because of the startup cost, accuracy, and standardization problems, editing is a key aspect of development. It is particularly important to verify the topological and coordinate accuracy of the file. Even if there were no developmental problems, "geographic" changes, such as new streets or changing area boundaries, make file editing essential to a useful GBF. The Census Bureau and related efforts have produced programs for off-line creation and batch editing of a DIME file.[12-13] These programs require a digitizer for data entry and take large amounts of computer and clerical time for editing. Although the procedures were used to create 200 GBF's, there has been little editing, and hence little use, of these files. Some cities (e.g., Reference 14) have developed their own GBF's similar to DIME. These efforts are also characterized by the use of a digitizer and batch computer programs for file creation, and by cumbersome file editing procedures. There have been a few efforts to develop on-line digitization systems, (e.g., Reference 15) and there are experimental systems which could support on-line digitization with visual feedback.[16-17] Yet, none of these systems provide all of the capabilities required for effective GBF creation and editing. Conclusions drawn from the IBM study[10] regarding the requirements for interactive GBF editing and maintenance were:

1. There must be a capability for projecting hard copy maps and/or photographs onto the display screen. It must be possible to select arbitrary (contiguous) sections of the maps, and to produce a range of scales.
2. The display system must be able to handle multiple, non-rectangular geographic coordinate systems.
3. The display system must be able to produce both

text and lines, with at least three colors for lines (in order to be able to distinguish two maps).

4. The display system must enable selection of any addressable point on the screen, whether or not anything is displayed at that point.
5. The creation and editing functions must include: digitization of base and overlay maps; labeling of points, lines, and polygons in the maps; moving and deleting points and lines; display of any section of the maps, and of specific points, lines and polygons; and checking for topological accuracy.

DATA EXTRACTION

Philosophy and operation

In the previous section, the combination of a GBF and properly structured source files containing geographic references were identified as the basis for offering a decision maker a comprehensive data base. Recent studies of interactive information systems applications in the solution of unstructured problems[4,18,19] have identified the need for reduced subsets of data for supporting the decision-making. Data reduction is required because:

a. the potentially useful data base will be much larger than the data actually used,
b. the user will want access to varying levels of detail in the data base,
c. the relevant subset of data will vary during the problem-solving process,
d. some data (e.g., census and event data such as police calls) may not be compatible at the detail level of the data captured in the source files.

Extraction is a process by which an integrated subset of data is developed from the source files relevant to a particular problem-solving application. Extraction thus provides the user with a capability effectively indistinguishable from a fully integrated data base, without requiring the development of such an integrated data base at the detail level of the source files, i.e., it provides a "virtually" integrated data base.

The extraction approach builds a data base subset from the source files according to a priori specifications for a particular application. Total integration of the source files, and dynamic aggregation and subsetting of the data at the time the data items are required is of course an alternative approach. This approach is not attractive in today's environment because:

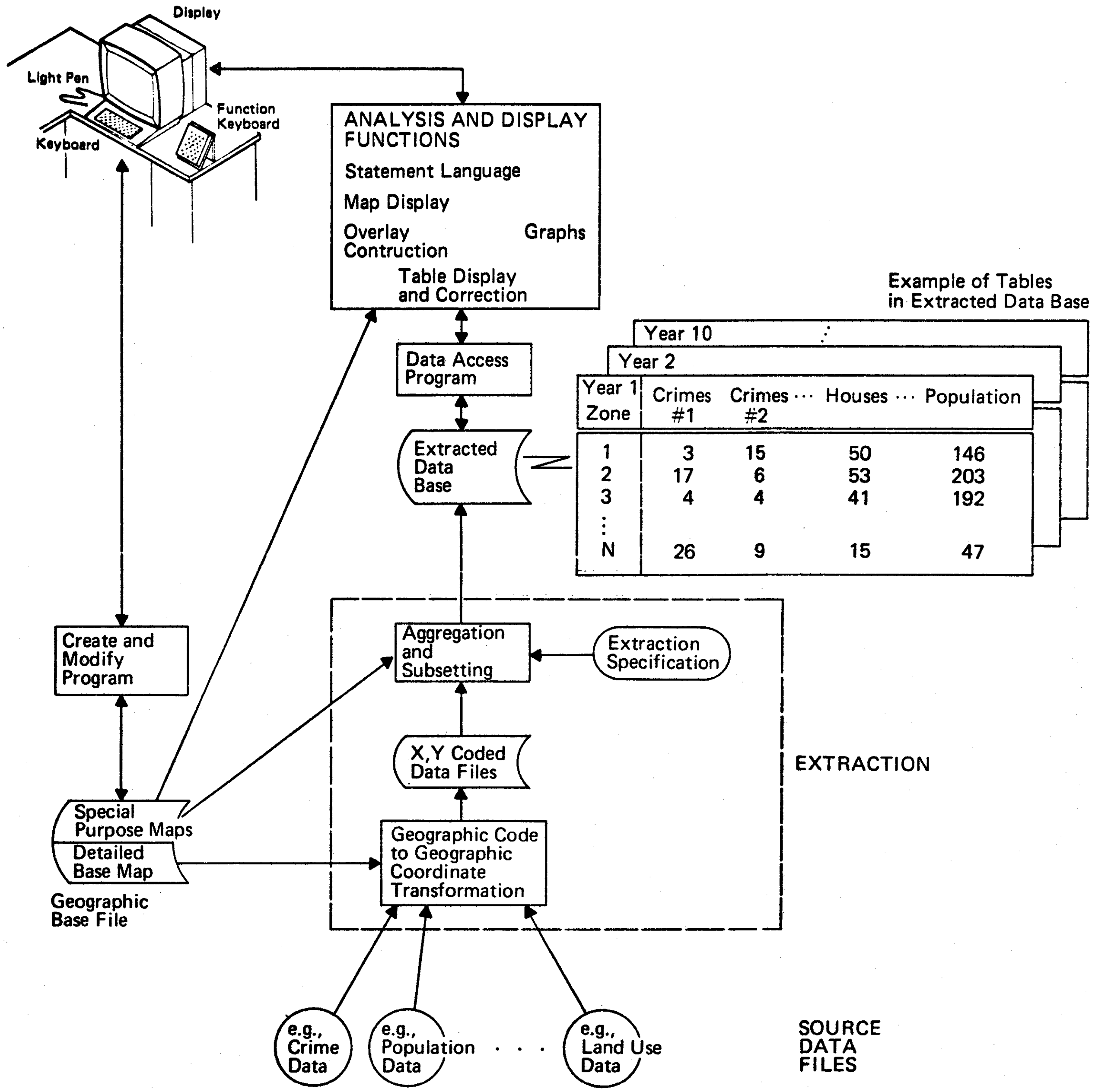
a. for any application all the relevant source files would have to be on-line to support conversational interaction,
b. protection of the source files would be more difficult,
c. development of conversational information systems would require additional, standardized data structures and codes for the dynamic aggregation and subsetting,
d. better conversational performance is possible when the problem solving accesses a smaller data base.

Clearly the development of a fully-integrated, on-line data base from the source files, solely for problem-solving applications, is not (currently) economical. Such an approach would also require special procedures for keeping the duplicate records current and consistent. With the extraction approach, the subset of data thought to be relevant to the particular problem is developed and made accessible to the problem-solving system in an extracted data base. The subset is an extract from the available source files at the level of detail desired by the decision maker for (that phase of) problem solving. This extracted data base may be thought of as a set of tables. Each table contains values for a set of variables extracted from the source files. For each variable there is one value in the table for each basic unit (e.g., zone, account, employee) used for the problem solving. New variables can be added directly to the extracted data base as an added column of the tables. An example of an extracted data base as implemented in the GADS system[3] is shown in Figure 1. The extracted data tables are formed from: source files containing 10 years' data on crimes, land use, and population; a special purpose map of police beat-building-blocks (basic zones); and an extraction specification for computing 20 crime categories and selecting population and number of houses by year. The result is 10 tables (one for each year) giving crime by category, population, and number of houses for each basic zone.

The extraction approach leaves control of the operational source files in the hands of the originating application. The extracted data bases are "snapshots" which are current at the time of their development. The problem solver can re-invoke the extraction process at any time to get a more current extracted data base. This process decouples the data base used in problem solving from the operational files, and assures the problem solver that the data base upon which he makes decisions is under his control. This user control of the extracted data base, and the potential performance advantages offered by access to the smaller extracted data set as compared to access to the total set of data, make the extraction approach attractive even in installations where an integrated data base exists. Extraction is simplified with the existence of an integrated data base, because there are then no difficulties with file formats and data conversion.

Extraction system architecture

The architecture of a municipal information system designed using the data extraction philosophy would have three major sets of programs and data bases (e.g., Figure 1). The first set would be the source data files and related programs for data entry, update, and other routine



Year 1 Zone	Crimes #1	Crimes ··· #2	Houses ···	Population
1	3	15	50	146
2	17	6	53	203
3	4	4	41	192
⋮				
N	26	9	15	47

Figure 1—GADS architecture

processing. These files should be "properly structured" as defined earlier in this paper. The data base management for these files may be an integrated system, such as IBM's IMS, or a more traditional system such as those provided by IBM's DOS. The second component includes accurate reference files (indices), such as the Geographic Base File, programs for maintaining these files, and programs for providing the data extraction functions of data matching, subsetting, and aggregation. This component is the key to integrating the data base of source files. The GADS experience indicates that it is possible to develop general purpose programs for the data extraction functions. Essentially, these functions provide integration through user-invoked processing, rather than through the complicated data structures and accompanying processing overhead often found in integrated data base systems. The data extraction programs are the interface between the municipal data base and the third component of the architecture, the extracted data bases and associated decision support system. The GADS analysis and display functions are an example of a decision support system for non-programmer users. A data extraction interface can provide multiple extracted data bases for a single decision support system, or for multiple decision support systems. For example there might be decision support systems for cash management, budget preparation, urban planning, computer-assisted appraisal, crime analysis, etc., all supported by a common extraction interface. The data management techniques for the extracted data bases should be tailored for each decision support system. However, the data access techniques may be the same as those provided for the source data files.

The details of the data extraction architecture and the implementation requirements are beyond the scope of this paper and there will be installation-specific comments. There is, however, one general requirement for any data extraction system. This requirement pertains to the data aggregation functions of extraction and can be described by considering examples of the data sources encountered in municipal governments and the kinds of extracted data to be developed from these sources. Consideration here is limited to data which can be related to points or areas. Data related to networks, budget items, part numbers, etc., should be handled in an analogous fashion.

1. Compatible data

This is the easiest, and fortunately the most frequent situation, if data files are "properly structured" as defined previously. The data in source files which can be identified with geographic points (x,y) can be directly related (and aggregated). If the extracted data base is to be relevant to a study of slum dwellings, for example, and if health cases, fire alarms and building code violations are all data sources which are available at the event level, (i.e., by address) then an extracted data table showing incidence of each of these events for specified addresses can be directly developed. Another frequently used extracted data base is the tabulation of such event data by geographical area, in terms of a specified map. (The extracted data base in Figure 1 is an example of this.) Extracted data bases in such cases are obtained by matching coordinates of events to the corresponding map areas (via point-in-polygon processing of the event coordinates against the map boundaries specification).

2. Non-compatible area data

If data is available by areas in the source files, and these areas are not compatible (i.e., one map is not a subset of the other), the extraction process is more complicated. For a chosen set of variables from the source files, there is a minimum level of aggregation at which an extracted data base is possible. For example, school attendance areas and police beats (and therefore the associated data) may only be compatible at the census tract level, i.e., they may both be (different) finer partitions of census tracts. The extraction process should alert the user to the non-compatibility and display for the user the minimum level of aggregation necessary for compatibility of the data sources of interest, in the form of a map, and permit the user to specify further aggregation from this map as desired.

If the user desires an extracted data base at a detail level finer than is compatible with the data sources given, the user must supply additional information. For example, suppose the user is studying property values vs. age distribution of inhabitants, with the age data on citizens available from the census only at census tract levels of aggregation. Compatibility exists at the census tract level. Any finer detailed extracted data base, at the city block level for example, could only be developed if the user is willing to make assumptions (such as homogeneity of the distribution of population ages in the census tract).

SUMMARY AND CONCLUSIONS

The development of information for decision-making in municipalities requires integration of data from the various operational files which are generated in local government. Even when an integrated municipal data base does not exist, it is possible to develop integrated data from properly structured source files in conjunction with a well-maintained reference file, such as a Geographic Base File. The current sources of information developed in municipalities, in particular the property data of the tax assessor function and the operating files of various service delivery functions, provide a rich source of information, augmented by special collections such as the U.S. Census.

Data Extraction is the process of developing integrated data subsets from diverse source files to support interactive problem solving. Extraction provides the interface between large data bases of source files and problem solving systems through data matching, subsetting and aggregation functions. Our experience with GADS has shown that data extraction is useful when the user or problem characteristics require access to varying amounts, detail, and selection of data, and conversational (rapid response) interaction with a problem solving system. These characteristics are likely to be encountered when designing problem solving systems for nonprogrammer, professional users working on unstructured problems. The data extraction interface matches the functional and response time requirements of interactive problem solving, can be implemented on a variety of computer system configurations, and can reduce the operating costs of the problem solving system.

Because data extraction operations can produce multiple extracted data bases, with different structures, a single data extraction interface can support multiple problem solving systems. In addition, existing problem solving systems can be supported and enhanced by data extraction without major program revisions.

REFERENCES

1. Mantey, P. E., J. L. Bennett, E. D. Carlson, "Information for Problem Solving: The Development of an Interactive Geographic Information System," *IEEE Int. Conf. on Communication, Vol. II.* Seattle, Wash. June 1973.

2. Cristiani, E. J., R. J. Evey, R. E. Goldman, P. E. Mantey, "An Interactive System for Aiding Evaluation of Local Government Policies," *IEEE Transactions on Systems, Man & Cybernetics*, Vol. SMC-3, No. 2, March 1973, pp. 141-146.
3. Carlson, E. D., J. L. Bennett, G. M. Giddings and P. E. Mantey, "The Design and Evaluation of an Interactive Geo-data Analysis and Display System," *Proceedings of the IFIP Congress 74*, International Federation for Information Processing, Stockholm, August 1974. North Holland Publishing Company, Amsterdam, 1974.
4. Carlson, E. D., and J. A. Sutton, *A Case Study of Non-programmer Interactive Problem-Solving*, IBM Research Report, RJ 1382, IBM Research Laboratory, San Jose, Ca., April 1974.
5. Hammer, T. R., R. E. Coughlin, E. T. Horn IV, "The Effect of a Large Urban Park on Real Estate Values," *Journal of the American Institute of Planners,* Vol. 40, No. 4, July 1974, pp. 274-277.
6. "City Hall's Approaching Revolution in Service Delivery," *Nation's Cities,* January 1972.
7. *Concepts of an Urban Management Information System*, a Report to the City of New Haven, Connecticut, by Advanced Systems Development Division, IBM Corporation, Yorktown, January 1967.
8. *A Municipal Information and Decision System*, University of Southern California, School of Public Administration, 1968.
9. Stickrod, R. L. and L. C. Martin, *Data Processing: Analysis of Costs, Benefits, and Resource Allocations*, Lane County, Oregon, Management Report, February, 1973.
10. Giddings, G. M. and E. D. Carlson, *An Interactive System for Creating, Editing and Displaying a Geographic Base File*, IBM Research Report, IBM Research Laboratory, San Jose, California, 1973.
11. Symons, D. C., *A Parcel Geocoding System for Urban and Rural Information*, Ottawa, Ontario, National Capital Commission, 1970.
12. U.S. Bureau of the Census, Census Use Study, *The DIME Geocoding System* Report No. 4, Washington, D.C., 1970.
13. U.S. Bureau of The Census, Census Use Study, *The DIME Editing System*, Washington, D.C. 1970.
14. Jull, R., *Geo-Modeling: A Local Approach*, Eugene, Oregon, Lane Council of Governments, 1972.
15. Hogan, R. D., *Remote Graphic Terminal and Urban Geographic Information System Demonstration*, Gaithersburg, Maryland, IBM Federal Systems Center, 1968.
16. Merrill, R. D., "Representation of contours and regions for efficient computer search," *Communications of the ACM*, Vol. 16, No. 2, (February 1973, pp. 69-82.
17. Saderholm, B. V., *Paper 'Keyboard' Runs Experimental IBM System*, IBM Research Division Press Release, Yorktown Heights, N.Y., March 8, 1973.
18. Cyert, R. M., H. A. Simon and D. B. Throw, "Observation of a business decision," *Journal of Business*, 29, 1956, 237-248.
19. Peace, D. M. S. and R. S. Easterby, "The evaluation of user interaction with computer-based management information systems," *Human Factors,* 15, April 1973, pp. 163-177.

Area Director:
Glen C. Bacon
IBM Corporation
San Jose, California

Storage technology

The sessions on storage provide a review of recent activity in mass storage along with an update on the several technologies which are contending for a place in the storage hierarchy. These two sessions are complemented by two sessions concerned with systems and reliability aspects. On the latter issue, the application of algebraic coding theory to both data transmission and data storage is investigated. An extensive piece of theoretical and experimental work is reported and hardware implications for mass storage are considered.

Very large mass storage devices have been the subject of increasing industry attention as a result of new activity by manufacturers. Particular attention is given to the internal architecture as well as the software aspects of these storage systems. Additionally, the session addresses the operational requirements for such devices in order that they may solve the major physical data hdndling problems in a large computer installation.

There is a high degree of activity in many industrial laboratories searching for better technologies than those which currently reside in a hierarchy. An entire session will be devoted to understanding the state of development of the novel technologies. These include bubbles, CCD's, super-conductors, holographic storage, and electron-beam storage. Each of these technologies offers different cost and system design trade-offs. Each has particular advantages which motivate its ultimate promise. The goal of this session is to give a comprehensive and balanced view of these, in order that their long-term promise can be assessed.

The last session is a panel addressing the system applications of advancing storage technology. As new storage function is provided and as old storage function becomes less costly, the systems architect has the opportunity to reassess his old trade-offs and consider new system structures. The panel will range from such issues as the opportunity for new technology in the access gap between memory and electromechanical storage to new system structure opportunities allowed by a very inexpensive memory removed from the central processor. Our goal is to produce a dialogue which will allow both the technologist and the system architect to better exploit the new memory technology.

Algebraic codes for improving the reliability of tape storage

by ELWYN R. BERLEKAMP

University of California at Berkeley and Cyclotomics, Incorporated
Berkeley, California

This paper describes an operational software package for protecting the integrity of tape files with the use of a sophisticated algebraic code, which provides a large amount of protection against a wide variety of possible types of errors for a relatively modest amount of redundancy. The software is implemented on a Univac 1108 computer. Each 36-bit computer word is treated as three 12-bit digits.

Because the encoder and decoder can work on bytes of 12 bits rather than on individual bits one by one, the implementation of the encoder for this code appears to be no more time-consuming than the encoder for the popular interleaved binary Hamming codes which are much weaker. Our software encodes and decodes using Galois field arithmetic, which is described in a later section of this paper.

All of this software is operational on the Univac 1108 computer under the Exec-8 operating system. There are three primary routines: ENCODE, DECODE, and CHECK. There is also a considerable number of subordinate routines which the primary routines call in order to perform their intended functions, but these subordinate routines are not intended to be called externally. Some of the subordinate routines are written in Fortran and others are written in the 1108 Assembly language.

The input to the Fortran subroutine ENCODE (A, K) is an array A of 36-bit words. The words A(1), A(2), . . . , A(K) are the inputs. If $1335 \leq K \leq 2668$, ENCODE will compute 62 words of redundancy and store them at locations $A(K+1)$, $A(K+2)$, . . . , $A(K+62)$. In no case will ENCODE make any changes in $A(1)$, . . . , $A(K)$, but it will overwrite $A(K+1)$, $A(K+2)$, . . . , $A(K+62)$.

The output of ENCODE is a block of N 36-bit words, which can also be viewed as $n=3N$ 12-bit *digits*. Each 12-bit digit may be considered as an element in the Galois field of order 2^{12}. With this interpretation, the output of ENCODE is a codeword in an appropriately modified reversible Reed-Solomon code, whose redundancy is 91 digits, rounded up to 31 words.

CHECK(A, N) is an integer function whose inputs are $A(1)$, $A(2)$, . . . , $A(N)$. If $A(1)$, . . . , $A(N)$ is identical to a block of words which was formerly the output of ENCODE, then the value of the integer function CHECK is zero. If the value of the interger function CHECK is nonzero, then its input is a garbled version of a previous output of ENCODE. It is possible (but very unlikely) that CHECK may have value zero even when its input is garbled. This event cannot occur unless the garbles affect at least 6 digits of the A block, and affect them in a very special (and extremely unlikely) way.

If CHECK returns a nonzero value, or even if CHECK returns zero but the user later encounters other difficulties which cause him to be suspicious of the accuracy of the A-block, then it is appropriate to call the integer function DECODE (A, N). The value of this function is an integer, e, which is a guaranteed lower bound on the number of garbled digits in the A-block. If DECODE returns a value of $e \leq 45$, then it has also changed and corrected the A-block into something which might have been the output of ENCODE. The corrected block is garble-free unless there actually were at least (91-e) or more garbled digits. If DECODE returns a value ≥ 46, then its value still serves as a lower bound on the number of garbled digits, but when DECODE detects more than 45 garbles, it is unable to locate and correct them, and so in this case it makes no changes in the A-block.

The high-level algorithms used by these programs are described in Berlekamp's *Algebraic Coding Theory,* McGraw-Hill, 1968.

RUNNING TIMES

All of our software is implemented in two versions. The faster version attains greater speed with the use of several large Read-Only tables. In situations where the memory space is limited, it is preferable to use the slower (and smaller) versions of our programs.

The running times of our programs, in milliseconds, for a block of K data words, $(1334 \leq K \leq 2668)$, are as follows:

Small version:

ENCODE	$t=1.3K+1460$
CHECK(,,4)	$t=.085K$
CHECK(,,6)	$t=.095K$
DECODE (10 errors)	$t=2.33K+4800$

Fast version:

ENCODE	$t=.44K+200$
CHECK(,,4)	$t=.028\mathrm{K}$
CHECK(,,6)	$t=.033K$
DECODE (10 errors)	$t=1.43+3250$

The running time of DECODE depends on the error pattern which it is correcting.

Since DECODE is called only on those rare blocks which are otherwise illegible, its running time is not a significant cost. The major operational cost is in running ENCODE on the many blocks of archival data which Census seeks to protect. For that reason, the rest of this paper will be devoted to our encoding programs, even though DECODE is much longer and substantially more complicated.

Galois field arithmetic

We have found a representation of the Galois field of order 4096 which is particularly well-suited for implementation on the Univac 1108 computer. To construct this representation, we first observe that the polynomial x^6+x+1 is irreducible over the binary field, but it has roots in the Galois field of order 64. Let ∂ be an element in the Galois field of order 64 which satisfies the equation $\partial^6+\partial+1=0$. Then the powers of ∂ include all 63 nonzero elements in the Galois field of order 64.

The quadratic $x^2+x+\beta$ is irreducible over the Galois field of order 64 if and only if $\mathrm{Tr}(\beta)\neq 0$, where $\mathrm{Tr}(y)$ is defined as $y^{2^0}+y^{2^1}+y^{2^2}+y^{2^3}+y^{2^4}+y^{2^5}$. Since $\mathrm{Tr}(\partial)=0$, the quadratic $x^2+x+\partial$ has two roots in the Galois field of order 64. However, since $\mathrm{Tr}(\partial^{-1})=1$, the quadratic $x^2+x+\partial^{-1}$ is irreducible over the Galois field of order 64, but it has two roots in the Galois field of order 4096. Let α be an element in the Galois field of order 4096 which satisfies the equation $\alpha^2+\alpha+\partial^{-1}=0$. We have determined that all 4095 nonzero elements in the Galois field of order 4096 are powers of α, and we have constructed tables of logs and antilogs (base α) for all such elements. We have also constructed short tables of logs and antilogs (base ∂) for the Galois field of order 64.

Any element in the Galois field of order 4096 can be uniquely represented in the form

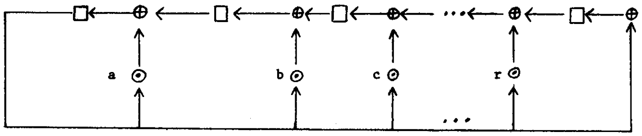
$$\sum_{i=0}^{5} A_1\partial^i+\alpha\sum_{i=6}^{11} A_{i-6}\partial^i$$

where $A_0, A_1, \ldots, A_{11}$ are in the subfield of order 2. This is the representation which we are using. The "real" part is in bits A_0, A_1, . . . , A_5; the "imaginary" part is in bits A_6, A_7, . . . , A_{11}. Each 36-bit Univac 1108 computer word is partitioned into three 12-bit digits, and each digit corresponds to an element in the Galois field of order 4096 according to the representation just explained. Our software is implemented in two versions. The fast versions perform multiplication and division by using log and antilog tables base α. These tables have length 4096, and the antilog table is duplicated to attain even greater speed.

The short (but slower) versions of our programs perform multiplication and division using log and antilog tables, base ∂, for the field of order 64. Addition in either field is accomplished directly via the Univac 1108 exclusive-or instruction. Multiplication in the field of order 4096 is reduced to a sequence of calculations in the field of order 64 by the formula $(A+B\alpha)(C+D\alpha)=(AC+\partial^{-1}BD)+(AD+BC+BD)\alpha$. Here A, B, C, and D all lie in the Galois field of order 64, where the arithmetic can be performed using the short tables. Division is accomplished by multiplying the inverse of the denominator. To find the inverse of $(A+B\alpha)$, we first compute its conjugate, $(A+B\alpha)^{64}$, and its norm, $(A+B\alpha)^{65}$. Since the norm lies in the Galois field of order 64, division by the norm involves only arithmetic in the smaller field, and the inverse of $(A+B\alpha)$ thus requires only the short tables.

ENCODING

The code which we use is a reversible form of a Reed-Solomon Code. It is a cyclic code. As described in Berlekamp's *Algebraic Coding Theory*, the encoder for a cyclic code of high rate is simply a shift register which divides the stream of information digits by the code's generator polynomial:



where a, b, c, . . . are r constant digits depending only on the code, and not on the information digits. They can be written into the program. The $\oplus$ is an exclusive *OR* operation. The. denotes a Galois field multiplication. Using a duplicate stored antilog table of total size 2×2048 and a stored log table of size 1024, it is possible on the Univac 1108 computer to simulate three shifts of any feedback shift register of degree r with an inner loop of only $3r+16$ instruction, each running at 750 nanoseconds, as long as r does not exceed 15. For example, if $a=\alpha^{73}$, $b=\alpha^{582}$, $c=\alpha^{417}, \ldots$, the inner loop of the encoding program is shown as in Appendix I.

If r (the number of shift register stages) exceeds 15, the Univac 1108 has too few accumulators for the above program to work.

The code which we use has generator polynomial of degree 91. Hence, the shorter version of our encoder represents this generator polynomial as the product of 13 factors, $g^{(1)}(x)$, $g^{(2)}(x), \ldots, g^{(13)}(x)$, where ten of the factors have degree 8; one factor has degree 6; another has degree 4; and one factor, $x+1$, has degree 1. The coefficients of each of these factors has been computed and compiled into fast assembly-language programs. In order to keep most of the encoding arithmetic in the Galois field of order 64, the shorter version of our encoder uses factors which also have conjugate roots, and therefore twice the degrees of the factors mentioned above. The biggest factors thus have degree 16.

The input to the encoder is considered to be the coefficients of a "message polynomial." Our encoding routines begin by dividing this polynomial by each of the above-mentioned factors of the generator polynomial. Since the degree of the divisor never exceeds 16, each division is quickly accomplished by a fast assembly-language program which treats the 16 a-registers of the Univac 1108 as a single shift register.

Let $r^{(i)}(x)$ denote the remainder of the message polynomial divided by $g^{(i)}(x)$. After computing $r^{(i)}(x)$, the encoder next obtains a corresponding "prenormalized" remainder, $t^{(i)}(x)$, by multiplying $r^{(i)}(x)$ by a precalculated polynomial $v^{(i)}(x)$ and then reducing modulo $g^{(i)}(x)$. The check digits, which are the coefficients of $r(x)$, the remainder if the message polynomial had been divided by the full generator polynomial $g(x)$, are then calculated via the formula $r(x) = \sum_i t^{(j)}(x)\prod_{j+i} g^{(j)}(x)$

The fact that this formula actually gives the correct values of the redundant digits depends on the Chinese remainder theorem and the fact that the pre-calculated $v^{(i)}(x)$ satisfy the equation

$$1 \equiv v^{(i)}(x)\prod_{j\neq i} g^{(j)}(x) \bmod g^{(i)}(x)$$

Since

$$r^{(i)}(x) \equiv t^{(i)}(x)\prod_{j\neq i} g^{(j)}(x) \bmod g^{(i)}(x)$$

it follows that

$$r(x) \equiv r^{(i)}(x) \bmod g^{(i)}(x)$$

for every $i=1, 2, \ldots, 13$. This shows that our fast encoding programs give the same values of the check digits that might be obtained more directly by the much slower process of dividing the message polynomial by the complete generator polynomial all at once.

APPENDIX I—Sketch of Main Loop of ENCODER

```
LOOP    XOR        A14,     ANTILG + 73,      X1
        XOR        A13,     ANTILG + 582,     X1
        XOR        A12,     ANTILG + 417,     X1
        . . .
        XOR        A1,      ANTILG + ?,       X1
        XOR, 5     A0,      data,             X2
        SA, 5      A15,     TEM
        LX         X0,      TEM
        LX         X1,      LOG, TEM
        LA, 5      A0,      TEM
        JZ         A0,      Bypass[1]
        XOR        A14,     ANTILG + 73       X1
        . . .
        XOR        A1,      ANTILG + ?,       X1
        XOR, 6     A0,      data,             X2
        SA, 5      A15,     TEM
        LX         X0,      TEM
        LX         X1,      LOG, TEM
        LA, 5      A0,      TEM
        JZ         A0,      Bypass[2]
        XOR        A14,     ANTILG = 73,      X1
        . . .
        XOR, 7     A0,      data,             X2
        SA, 5      A15,     TEM
        . . .
        LA, 5      A0,      TEM
        JZ         A0,      Bypass[3]
        JMGI       X2,      LOOP
```

Bridging the memory access gap

by DENNIS E. SPELIOTIS
Micro-Bit Corporation
Lexington, Massachusetts

INTRODUCTION

The rapid growth of electronic data processing over the past two decades has been characterized by an almost insatiable appetite for larger and faster memories. In fact, over this period of time, on-line storage capacity has increased about three times as much as CPU power.[1] Yet, in spite of this impressive growth rate, memory represents perhaps the most limiting area in the development of more advanced computer systems. Furthermore, the increasing diversity of storage devices and the wide disparity in the price-performance of these devices, present a difficult challenge to the system designer, and account for a large part of the complexity of the software and hardware system to manage the storage facilities.

Viewed simplistically, memory devices can be classified into two basic categories: electronically accessed main memory and electromechanically accessed peripheral memory. The former is fast and relatively expensive, while the latter is very slow (typically a factor of 10^4 to 10^5 slower access) and relatively inexpensive (by about a factor of 10^{-1} to 10^{-3} in price per bit). Between these vastly separated device technologies we have the famous memory access gap,[1] which has persisted essentially unchanged over the last twenty years even though the boundaries on either side of the gap have moved toward faster access by about an order of magnitude over this same period of time.

The absence of a bridging technology is in no way the result of a lack of effort and intensive search to develop such a technology. Suffice it to mention cryogenics, thin magnetic films, thermoplastics, and magneto-optics as a partial list of the most spectacular but unfruitful endeavors in this domain. In recent years we find a tremendous effort concentrated on magnetic bubbles and CCD (charge coupled devices). However, the technology with the greatest promise and potential in bridging the access gap is the three quarters of a century old electron beam, which ironically also provided the access means for the very first random access memory way back when it all began.[2]

The purpose of this paper is to set forth the fundamental arguments on "why electron beams", then to describe the particular desirable attributes of an electron beam addressable memory system, and finally to describe the achievements to date and the expectations for the future.

WHY ELECTRON BEAMS

Peripheral memories

Peripheral memories in digital computer systems have been dominated by magnetic recording devices, such as disks, drums, and tapes. The spectacular success and growth of magnetic recording storage devices derive from certain inherent characteristics of the technology:

(a) Very low cost storage media based on homogeneous (non-discrete) magnetic surfaces.
(b) A means of accessing which allows tens of millions of bits to share one write-read transducer and encode-decode-sense channel.
(c) The fundamental limits of the technology were far beyond the demands placed on the technology, thus allowing plenty of room for growth and expansion in device capability and performance. Using areal density as a measure of device sophistication, disk systems for example have gone from 2.2×10^3 bits/in^2 in RAMAC I introduced in 1956 to 2.24×10^6 bits/in^2 in the CDC 9762 introduced in 1974, a factor of one thousand improvement in packing density!

The fundamental deficiency of disks is their slow electro-mechanical access. System designers have resorted to a number of techniques in order to partially mask the long access time, but all these approaches are costly and less than satisfactory at best. They include:

(a) Queuing and look-ahead to minimize disk arm motion. This is difficult to optimize in multiprocessing environments and at high priority interrupt frequencies.
(b) Transfer large blocks to minimize the frequency of accesses to the peripheral device. This requires expensive buffering, and depending on the data transfer bandwidth could actually result in lengthening of the average access time in certain processing environments.
(c) Employ fixed head disks and drums to eliminate arm motion and head positioning time. This reduces the average access time by a factor of three to five,

but increases the per bit price by one to two orders of magnitude.

(d) Employ more than one head per arm, like in the IBM Winchester disk system, or more than one head per track, like in the IBM 2305-I disk system—both of which are rather expensive propositions for only a small improvement in access time.

It is clearly evident that the slow access time of the electromechanically-accessed peripheral memories is a very serious bottleneck in improving system performance, and the techniques that are being used to shorten it provide only small relief at considerable cost. Furthermore, the evolution of computing systems toward timesharing, virtual storage, multiprocessing, and network processing tends to aggravate the situation and places additional emphasis for a technology that bridges the access gap. Until such a technology becomes available, we will continue to see an accelerated growth in the size of main memory—and a corresponding increase in total system cost—as a necessary prerequisite for efficient system performance.

Main memories

Main memories on the other hand have price-performance characteristics which are essentially the opposite of peripheral memories. Their outstanding advantage is their very fast access time achieved by direct wired access to each bit. Their disadvantage is the high cost per bit, and the reasons for this are several:

(a) The bits are physically discrete entities. This has very important reflections on the cost to introduce the discreteness and obtain satisfactory yields.

(b) Wired access to each bit also reflects on the cost to introduce or install the wiring, and to make the many thousands of interconnections needed for a sizable memory. It also affects the yield and the reliability of the devices, particularly as the bit packing density and the total capacity of the memory increase.

(c) The number of bits that can share a sense amplifier is typically a few thousand as contrasted to tens of millions in the case of peripheral memories.

It can be argued that the cost of main memory will continue to decrease as more integration and automation are introduced into device fabrication. However, the above arguments still apply, and diminishing returns will tend to dampen cost improvements, particularly for mature technologies.

Searching for a gap-filling technology

Ideally what we need is a new technology which can approach on the one hand the access time of main memories, and on the other the per bit price of peripheral memories. Clearly, such a technology must employ electronic accessing. It should also incorporate many of the other attributes which contribute to the low per bit cost of peripheral memories, such as high bit packing density ($\geq 10^6$ bits per square inch), avoid structure or discreteness for defining the bits, employ a minimum number of interconnections, and allow for the sharing of a very large number of bits by a sense amplifier. Many different schemes have been tried and a much greater number of techniques have been proposed for a memory technology which may satisfy the above requirements. Generically they fit into two categories with different philosophy of accessing:[3]

Moving the bits to the sensor

This category includes all the shift register types of devices which electronically propagate one or more series of bits to a sense amplifier through a fixed propagating structure. The prime examples of such technologies are charge coupled devices (CCD) and magnetic bubble memories, the latter of which in particular is receiving a tremendous amount of attention currently.

If we examine the potential of bubble memories on the basis of the criteria outlined above for peripheral and main memories, we can make the following observations:

(a) Even though the storage medium is homogeneous, the propagating structure is not.

(b) Material perfection requirements in moving bit devices are tough, and this reflects on yields and cost.

(c) Very small bubbles which would permit high bit packing densities have been observed only in amorphous films and bubble lattice structures, both of which have their own materials and processing complexities.

(d) Bubble propagation speeds are rather slow, and even though paralleling is straightforward, it adds rapidly to the cost of driver-sense electronics.

(e) Major-minor loop organizations to facilitate the sharing of a large number of bits by one sense amplifier in order to reduce cost tend to degrade their access time.

The most significant advantages of this technology are non-volatility, low power requirements, and volumetric compactness.

The above observations on magnetic bubbles essentially apply also to the CCD technology, but with some significant differences. The access time and propagation time of CCD is faster by about one order of magnitude than that of magnetic bubbles. Offsetting this advantage are certain disadvantages, which include volatility and the need for very frequent refreshes of the data pattern (a factor which can have significant repercussions on error rates even when the memory is not being accessed), higher power requirements, and the need for a large amount of additional electronic circuitry for selecting, driving, and refreshing the much shorter data loops.

Clearly, for both CCD and magnetic bubbles, the fabrication and processing complexity, the level of discreteness in the definition of the bits, the areal bit density, the number of interconnections, and the number of bits that can share one write-read channel, fall in the intermediate area between main memory and disk-type peripherals, as does their access time. Therefore, their price-performance potential would place them in the "classical" access gap, where they should offer cost competitiveness with small (up to 20 megabits) fixed-head disk or drum systems, but with an all solid-state technology and with significant improvements in performance. More specifically, we would expect CCD to penetrate the small auxiliary storage sector where performance is paramount, whereas magnetic bubble memories will be used in special applications where moderate performance is acceptable, but ruggedness, reliability, non-volatility, compactness, and low power consumption are emphasized (aerospace systems, process control, word processing, numerical control, and telecommunications).

Moving the sensor to the bits

In this category we place technologies that employ an inertialess access mechanism which interacts with a stationary storage medium for the writing and reading of information. Consequently, our discussion is restricted to devices which employ high energy beams of either sound, light, or electrons as the accessing mechanism.

Sonic beams

Fundamental limitations in focusing and deflecting a sonic beam preclude the feasibility of using it as a random access addressing mechanism. Serial access utilizing a magnetostrictive film as the storage medium has been demonstrated,[4] and even though the transfer rates are very attractive the bit densities are not, which would tend to exclude this approach as a serious contender for general purpose beam addressable memory applications.

Light beams

Spurred on by the development of the laser and holography, a tremendous amount of attention has been given to the development of optically accessed memories. Unfortunately today, more than a decade later, the prospects appear less than exciting for various reasons the most important of which are:

(a) The most fundamental problem is the development of a suitable, nonvolatile, erasable, optical storage medium. Many different materials and interaction modes have been investigated, including magneto-optics,[5,6] thermoplastics, photochromics, photodichromics, electro-optics, and amorphous semiconductors. All of these storage materials have one or more basic shortcomings which limit their application and usefulness (such as low sensitivity requiring enormous beam energy densities for writing and erasing particularly for holograms, limited reversibility, low diffraction efficiency, need for cryogenic operating temperatures).

(b) The other fundamental problem is the development of high speed, high repetition rate, low cost digital deflectors which can address a large number of resolution elements. Because of the overhead required for the generation, modulation, focusing and deflection of a light beam, the minimum capacity for an economical memory system would have to be about 10^8 bits. For bit by bit recording, this would require a deflector system capable of accessing $10^4 \times 10^4$ resolvable spots, a requirement far exceeding the capabilities of acousto-optical and electro-optical deflectors. Such deflectors are adequate for a page organized holographic memory, but this approach is impractical today due to the unavailability of suitable materials (except for read-only photographic emulsions) and page composers[7] for input data formatting prior to exposure of the holograms.

(c) Even if the deflector limitations to bit by bit recording were to be eliminated through some breakthrough in development, still the addressability of a field of 10^8 bits would present formidable problems due to diffraction, depth of field, depth of focus, aberrations in the optical components, and accuracy and stability requirements in the deflection electronics. Proposals[6] to get around these problems by incorporating mechanical motion of the storage medium are unattractive, because the achievable areal storage density could only be slightly higher than for magnetic recording while the performance would be comparable and the cost much higher.

Electron beams

Electron beams possess very attractive properties which render them far superior to light as a memory access mechanism.

(a) Like light, they can be formed into high energy density, high resolution beams, but since their diffraction limit is several orders of magnitude beyond that of visible light they are inherently capable of much greater resolution, depth of focus, and depth of field.

(b) Unlike light beams, deflection, modulation, and scanning of electron beams is exceptionally simple and fast.

(c) Because electron beams are strongly interactive with both electric and magnetic fields, we can envision a variety of materials as a possible storage medium, including ferroelectric, magnetic, semiconductor, thermoplastic and insulator films.

(d) Achievable spot size and energy density (current density) into the spot are closely interrelated, and are ultimately limited by the brightness of the source, the physical size of the electron-optical components, and the aberrations introduced by these components—particularly the deflector. On the other hand, the limits on relocating the spots are imposed by the accuracy and stability of the deflection electronics and by thermal and mechanical considerations. These constraints combine to set some practical limits on the number of spots that can be randomly addressed reliably in a single lens-deflector field, and that number is 10^7-10^8. Brighter sources such as field emitters would certainly allow much higher current into the spot, but would not appreciably alter this limit until more accurate deflection electronics became available. Even with thermal electron sources, however, we can conservatively project several million bits for a single lens-deflector field and available materials, corresponding to areal densities of 2×10^7-7×10^7 bits per square inch, which are indeed very impressive.

(e) The total field that can be accessed by a single electron beam can be expanded by several orders of magnitude by employing two stage deflection and an array of lenses known as the fly's eye[8] configuration. This approach removes the constraints imposed by deflection electronics and opens the road to the development of memories with capacities of several gigabits, access time of a few microseconds, and costs comparable to those of large disk files. It is precisely this unlimited potential and the tremendous versatility and capability of electron beams that makes them the most powerful developmental technology in the race to bridge the access gap.

DESIGN CRITERIA FOR AN ELECTRON BEAM MEMORY

In this section we discuss the components of an electron beam addressable memory (EBAM) system, the available choices in the design and configuration of the system, and some of the reasons for selecting a preferred embodiment in the practical implementation of the system. The discussion is aimed toward the general concepts of the design rather than the specific analytic details.

Electron optics

The electron source should be a dispenser-type cathode, which has long life of up to 50,000 hours at loadings of over 1 A/cm^2, and is readily available and inexpensive. Such cathodes incorporated into well designed guns operating at moderate beam energies (about 10 kV) can provide excellent brightness. Much higher brightness can be obtained with field emission cathodes which, however, need additional engineering development. Since brightness increases rapidly with increasing beam voltage, the selection of that voltage must be made as a compromise between brightness, operational requirements of the storage target, and the complexity and cost for insulation, power supplies, and deflection amplifiers, which increase with increasing voltage. A good choice would appear to be a beam voltage of about 10 kV.

For beam focusing and deflection, electrostatics is definitely a clear choice over magnetics for practical as well as fundamental reasons. Rapid random access requirements would exclude magnetic structures due to hysteretic or inductive limitations. Cost, size, and weight would also favor the electrostatic approach. Furthermore, the difficulty of confining magnetic fields would inhibit the close packing of a cluster of tubes in a system sharing power supplies and deflection electronics—a highly desirable configuration to reduce system cost and increase throughput bandwidth. Finally, magnetic focusing and deflection would be totally inapplicable for array electron-optical structures of the fly's eye type.

Storage medium

The storage target must be a stable material, compatible with high vacuum and bakeable to at least 350°C, possess some physical property which can be reversibly, rapidly, and efficiently altered by a high resolution electron beam, be capable of good "one"-"zero" discrimination and large signal-to noise ratio at very high bit packing densities, and have a long life under continuous operation. A crucial additional requirement is that the target be homogeneous and structureless in a plane perpendicular to the beam axis so that the bits are located at the points of beam incidence, rather than for the beam having to find predetermined bit locations related to a specific structure in the target plane. This requirement is imposed not only by cost considerations in introducing the structure and the resultant yields, but also by the previously mentioned need to share electronics for a matrix of tubes, and to accommodate some inherent residual aberrations and differences among tubes.

The lack of a suitable target that satisfies the above requirements has certainly been the most constraining limitation in the development of EBAM systems. Thermoplastic materials[9] have very limited reversibility due to polymerization induced by cross-linking. Similarly limited are amorphous to crystalline phase transitions in chalcogenide glasses.[10] A comparison between magnetic and electrostatic storage targets results strongly in favor of the latter. Lorentz interactions either with the magnetization or with the external fringing field of magnetic films are so weak as to preclude readout from such films at anywhere near the desired bit densities and speeds.[11] Also, if readout was based on using the energy of the beam to thermally disturb the cooperative coupling of assemblies of dipoles, electrostatics would again be favored over magnetics because of higher energy density. Ferroelectric thin films[12] are potentially powerful contenders for storage targets in EBAM systems, but additional materials development is required to fully realize their potential. Even tougher materials problems have impeded progress toward the development of a photoconductor-ferroelectric sandwich target.[13]

A material which satisfies very well the requirements for an EBAM storage medium is the silicon-silicon dioxide system, and it is indeed an extraordinary phenomenon that all known current approaches[14–17] to write/read electron beam memories are based in some way or another on silicon technology and electrostatic charge storage. Some of these approaches use surface charge storage in a finely etched structure on the oxide which is selectively charged by the beam for writing and serves to modulate either the secondary electron emission[14] or the transmissivity of a low energy beam[15] for reading; other approaches[16,17] use beam induced imbedded charge storage in the oxide and near the silicon dioxide-silicon interface, which modulates the depletion region in the silicon and serves to separate the beam induced charge carriers in the silicon for read out. The latter approaches are superior in that they employ structureless targets (for planes perpendicular to the beam axis), use beam voltages more compatible with high gun brightness and resolution, are directly usable in array optical configurations, and provide large local amplification during the reading and the writing operations. Even though such targets have their problems and limitations, they are based on a wide ranging and fast advancing technology, which can confidently provide the needed improvements. A more detailed description of the imbedded charge storage mechanism is given in the last section of this paper.

Other considerations

We have already mentioned the importance of sharing some of the electronics overhead among several tubes in order to amortize costs over a large storage capacity. Random access electron-optical memories require very stable and highly regulated power supplies, and a very fast and accurate deflection electronics system, both of which contribute in very significant proportion to the overall system cost, and hence the need for sharing. Using common deflection for a number of parallel channels (like 18), also results in the added advantage of greatly increasing system throughput and bandwidth. This, of course, implies that there exists a minimum total capacity below which electron optical memories are less cost effective, and this minimum is in the range of two to eight megabytes depending on whether the systems are used as main memory extensions or as high performance peripherals.

The access time of a few microseconds reflects primarily the deflection settling time. Consequently, even though EBAM systems can be used to access a single bit at a time, a more efficient utilization would result from a block addressable organization of the memory, where the beam settling delay is incurred only for the starting location of the block. Block organization also offers the added advantage of data encoding, and permits the reading of transitions rather than absolute charge levels, thus relaxing the requirements on signal uniformity.

The demise of the Williams tube and the subsequent slow progress in the development of electron beam memories was due largely to the basic deficiencies of the inadequately developed underlying technologies that are used in the making of such memories, and the unavailability until much later of the very large computers that are needed for design simulation of off-axis high resolution electron optics. Within the last few years, however, the confluence of significant advances and improvements in all the supporting technologies have made the realization of large and superior EBAM systems possible. These include:

(a) precise, stable, fast, and inexpensive electronic components
(b) silicon-based targets of excellent perfection and processing control
(c) long life dispenser-type cathodes and superior electron optical components
(d) advanced materials, fabrication techniques for low cost, high precision parts and assemblies, superior cleaning and handling techniques, and improved vacuum technology.

THE MOS ELECTRON BEAM MEMORY

Electron beams and MOS targets can be combined to create a new and powerful memory technology. The basic operation of the memory is illustrated in Figure 1. Information is stored in a pattern of positive charge in the

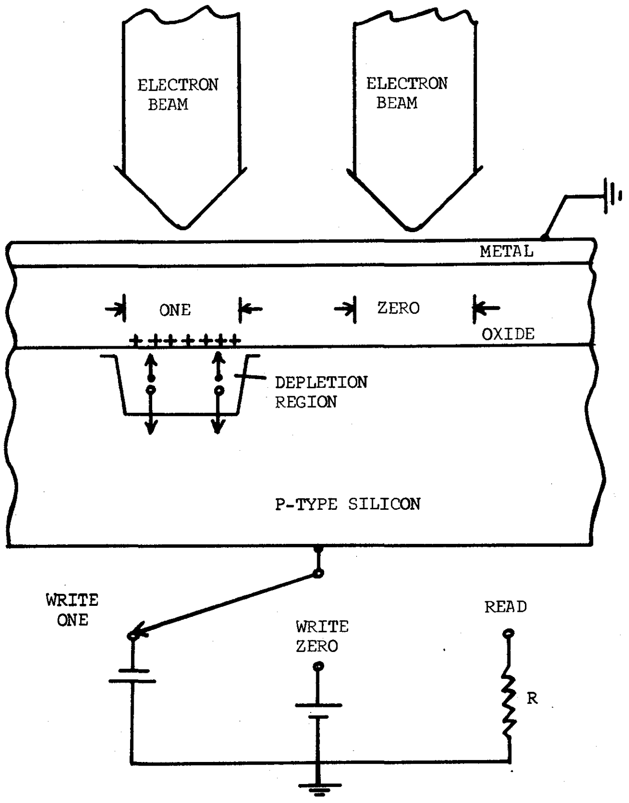


Figure 1—Storage and readout mechanism of the MOS electron beam memory (not to scale)

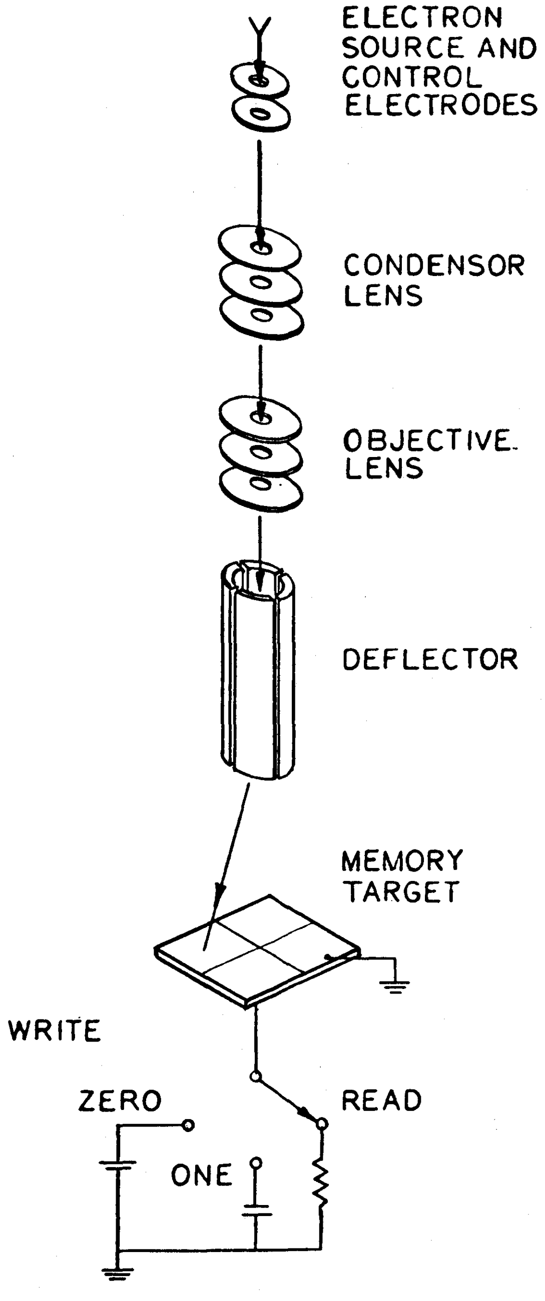


Figure 2—Schematic description of electron beam memory tube using single channel optics

oxide, near the interface to the semiconductor. The storage of positive charge corresponding to a "ONE" is accomplished by exposure to the beam under positive gate bias, while the removal of positive charge corresponding to a "ZERO" is accomplished by exposure to the beam under negative gate bias. The stored positive charge depletes the underlying regions of the semiconductor and drives the surface of the p-type substrate into inversion.

The energy of the beam (10 kV) and the thickness of the gate and the oxide are such that the beam penetrates several thousand Angstrons into the silicon. The reading of the stored information is accomplished in the silicon by using the same beam as for writing. The electron-hole pairs generated by the penetrating read beam in the semiconductor are separated by the strong field in the depleted regions under the ONES and a current is detected in the sensing resistor R, while the absence of a field under the ZEROS permits most of the generated charge carriers to recombine. Since the creation of an electron-hole pair requires an energy loss of about 3.7 eV, and the beam enters the semiconductor with an energy of a few keV, the readout signal is greatly amplified over the

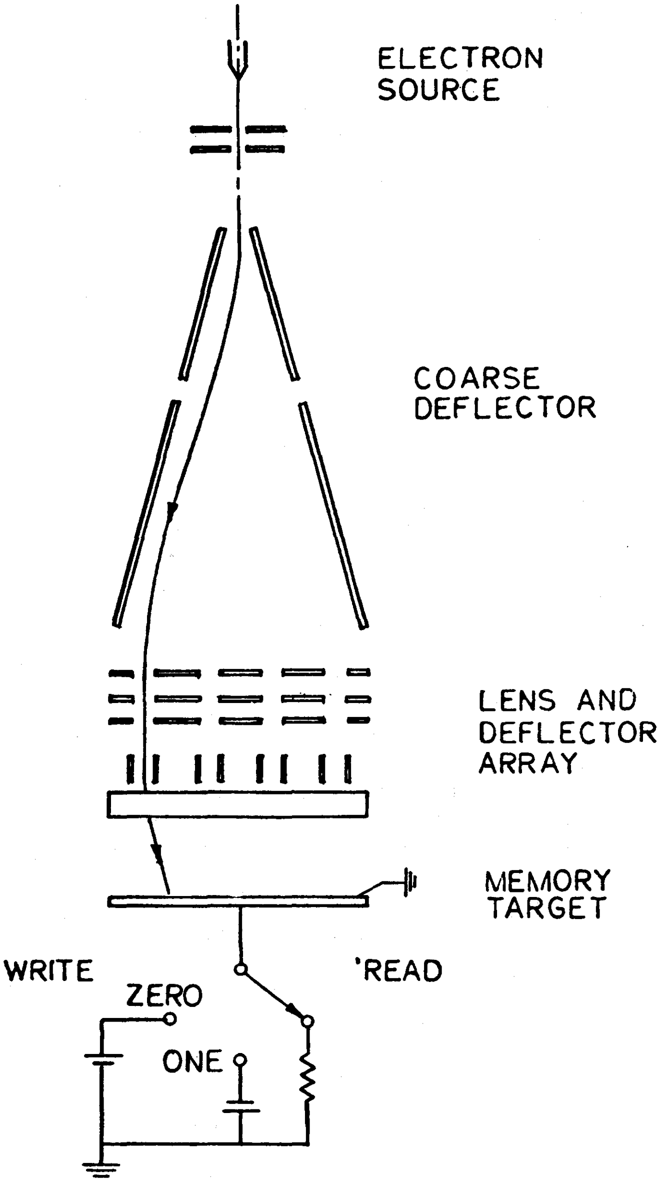


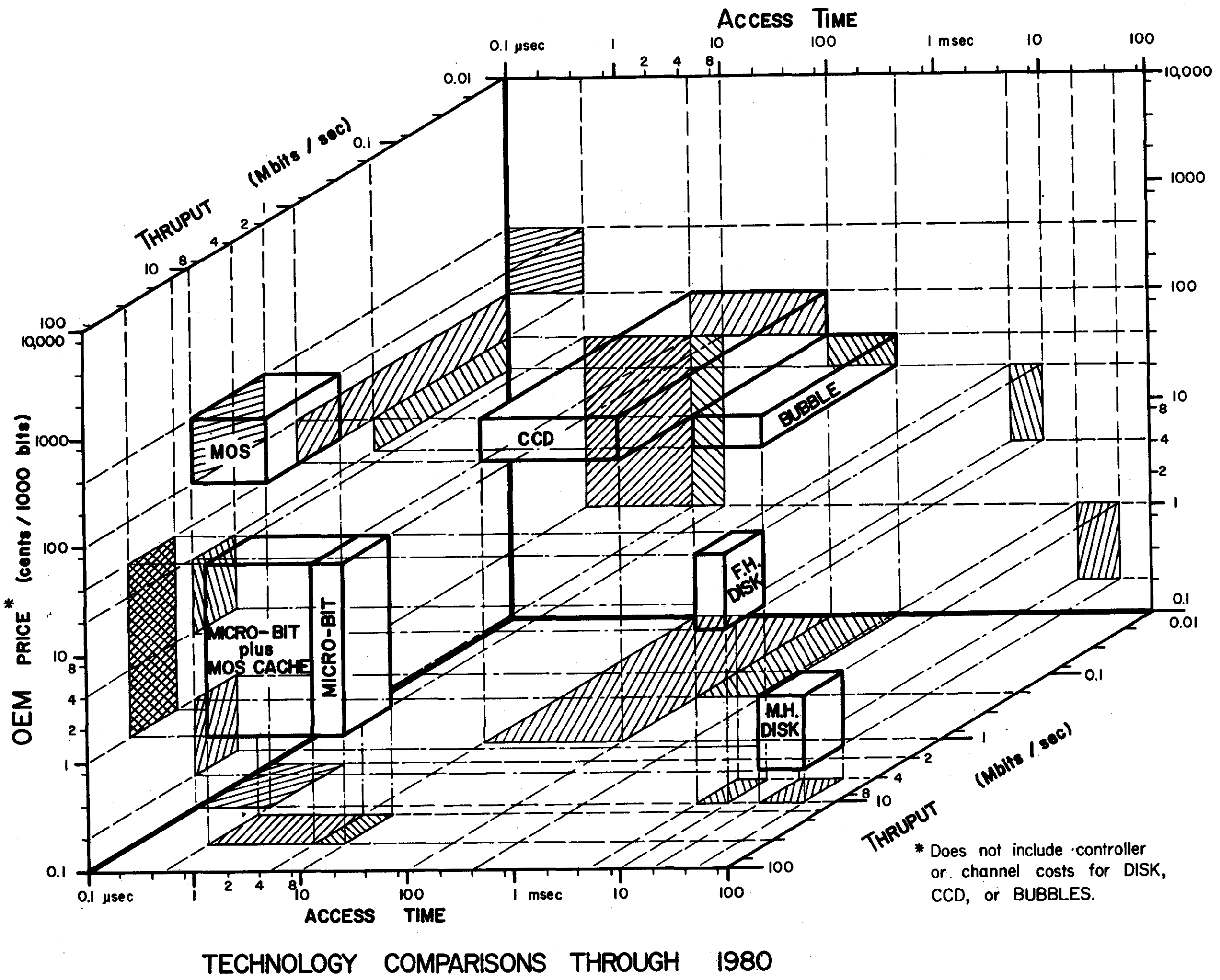
Figure 3—Schematic description of electron beam memory tube using array optics

read beam current by a large local and noiseless gain; a similar but much smaller gain also occurs during the writing process. This gain represents one of the most outstanding advantages of the MOS storage medium.

A schematic description of an EBAM tube under development using single channel optics is shown in Figure 2. Such a tube stores 4.2 megabits in 1 cm^2 target, using a 30 nA beam into a 2.5 micron spot, and is limited principally by deflection electronics accuracy and stability and by deflector aberrations. A memory system consisting of 18 such channels in parallel will have a capacity of 75 megabits, access time to any block of under 10 microseconds, and read/write throughput rates of 38 and 5 megabits/sec, respectively.

The limitations on the number of bits per tube imposed by deflection can be removed by using two stage deflection in the array optics configuration shown in Figure 3. This approach can start with tubes of 32 megabits capacity while the upper limits may be in the range of 0.5 to 5 gigabits per tube corresponding to bit densities of the order of 10^8 to 10^9 bits/cm^2! Because array optics also delivers a much higher current density into a given spot on the target, the throughput of the memory will also be considerably faster. A memory system consisting of 18 parallel channels of 32 megabits each, will have a capacity of about 600 megabits, access time to any block of under 12 microseconds, and read/write throughput rates of 90 and 40 megabits/sec, respectively.

Electron beam memories will initially be cost-competitive with fast auxiliary storage devices, and eventually with all on-line random access peripheral memories, but with far superior performance. Equally important, however, is their potential use as main memory extensions, in combination with a semiconductor cache, where they will have a large price advantage at comparable performance. This application is envisioned in Figure 4, which shows price-performance comparisons for EBAM and other memory technologies through the end of this decade.



* Does not include controller or channel costs for DISK, CCD, or BUBBLES.

Figure 4—Price-performance projections of various memory technologies through 1980. Transfer rates for the shift register devices are along each track

Electron beam memories have the power not only to occupy a strong place in the memory access gap, but also they have the potential to completely eliminate the gap and bring about a drastic simplification in system architecture and a large improvement in cost-performance.

ACKNOWLEDGMENT

It is a pleasure to acknowledge the many helpful discussions with my colleagues K. J. Harte and D. O. Smith.

REFERENCES

1. Pugh, E. W., "*IEEE Trans. Magnetics*, MAG-7, 810, 1971.
2. Williams, F. C., and T. Kilburn, *Proc. IEE* (London) Part III, 96, 81, 1949.
3. Smith, D. O., *Ann. N.Y. Acad. Sci.*, 189, 298, 1972.
4. Shahbender, R., R. Jerkart, H. Kurlansik and L. Onyshkevych, *IEEE Trans. Magnetics*, MAG-5, 427, 1969.
5. Chen, D., J. F. Ready, and E. Bernal G., *J. Appl. Phys.*, 39, 3916, 1968.
6. Eschenfelder, A. H., *J. Appl. Phys.*,41, 1372, 1970.
7. Rajchman, J. A., *J. Appl. Phys.*,41, 1376, 1970.
8. Newberry, S. P., T. H. Klotz, Jr., and E. C. Buschmann, *Proc. Nat. Electrn. Conf.*,23, 746, 1967.
9. Glenn, W. E., *J. Soc. Motion Picture Television Engrs.*, 69, 577, 1960.
10. Chen, A. C. M., *IEEE Trans. Electrn. Dev.*,ED-20, 160, 1973.
11. Cohen, M. S., *IEEE Trans. Mag.*,MAG-4, 639, 1968.
12. Smith, D. O., K. J. Harte, M. S. Cohen, S. P. Newberry and D. E. Speliotis, U.S. Patent No. 3,710,352, January 9, 1973.
13. Chapman, D. W., *Proc. IEEE Int. Computer Group Conf.*,56, 1970.
14. Kelly, J., J. S. Moore and P. R. Thornton, *NAECON '74 Record*, 55, 1974.
15. Heiman, R. and A. Waxman, *Digest IEEE Intl. Conv.*, New York, 152, 1970.
16. Cohen, M. S. and J. S. Moore, *J. Appl. Phys.*, 45, 5335, 1974.
17. Ellis, G. W., G. E. Possin and R. H. Wilson, *Appl. Phys. Let.*, 24, 419, 1974.

IBM 3850—Mass storage system

by CLAYTON JOHNSON
IBM Corporation
Boulder, Colorado

SUMMARY

IBM's 3850, a hierarchical storage system, provides random access to stored data with capacity ranging from 35×10^9 to 472×10^9 bytes. The hierarchical architecture achieves access times varying from Direct Access Storage Device speeds to that of the Mass Storage Facility which can be as low as 10 seconds. The architecture of the Mass Storage System is examined to demonstrate its functional and performance capability.

INTRODUCTION

The goal of the Mass Storage System (MSS) is to provide capacities for handling all of an establishment's data. In addition, it provides the functions of Direct Access Storage Devices (DASD) at a cost which approximates that of today's half-inch tape systems, and with an architecture that allows users to migrate to MSS with minimal costs.

A hierarchy of devices provides the flexibility needed to approach such a diverse set of goals. See Figure 1.

A hierarchical arrangement allows the MSS architecture to be hidden, or buffered, from main memory. This allows system designs that are specific to the problem of storage and retrieval of data, independent of how programs manipulate data. Utilization of an existing device eliminates the need to develop new Direct Access Devices (adding to the complexity of the development project) and provides a known, existing interface to data stored in the MSS. Actually, the intermediate devices in a storage hierarchy can be independent of the architectural interface to data. For example, in MSS the interface to data is architecturally that of an IBM 3336 Model 1 DASD volume, regardless of whether a 3330 Model 1, or Model 11 is being used. A new device or memory could be used in the intermediate levels of the hierarchy without changing the architectural interface to data itself. This allows for the incorporation of new technologies in the storage hierarchy with little or no disruptive impact to the users of the storage hierarchy.

The ideal control of the external storage is to have on DASD that data required by the programs in the CPU when the programs need it, and to have space available on DASD to store new data. How close we come to this ideal is a measure of the trade-off of price versus performance. In this hierarchy, the best performance achievable comes from DASD. The time a program waits for data to appear on DASD or waits for space on DASD to store new data, is the visible impact of the storage hierarchy.

A storage hierarchy also reduces the number of requests for data that must be resolved at the Mass Storage Facility. Recently active data sets can be kept on the DASD level of the hierarchy for a period of time after their last use. This is accomplished by providing storage capacity at the DASD level that is greater than the sum of the size of all concurrently active data sets.

Reuse of data dramatically reduces the number of accesses to the Mass Storage Facility, and makes it possible to design a system that moves much less data between MSF and DASD than that which moves between Main Memory and DASD.

MSS uses a Least Recently Used replacement algorithm to schedule which data on DASD should be destaged to the MSF when space is needed for new data, or when newly active data must be staged to DASD for access by the CPU(s). Only those DASD cylinders which have changed are actually destaged.

The following exemplifies the work load an MSS might experience, and the advantages of this architecture.

CONTROL AND DATA FLOW WITHIN MSS

MSS, as seen by the System Control Program, has nearly unlimited Direct Access Storage. This approach to Mass Storage furnishes the function of DASD, and it also provides a major new capability on IBM CPU systems without the disruptive impact of a totally new architecture. MSS also follows the basic architecture of System/370 channels and I/O addressing. The result avoids a major redesign of Operating Systems, User Application Programs and Channels.

Today's systems gain addressability to data or storage space by mounting a storage volume (either half-inch tape or disk packs) on an auxiliary storage device. These devices have specific addresses on one or more CPU channels. An Input or Output operation is performed to a specific volume by addressing the specific device holding that volume. The total addressable data at any one time is limited in this arrangement to the actual number of

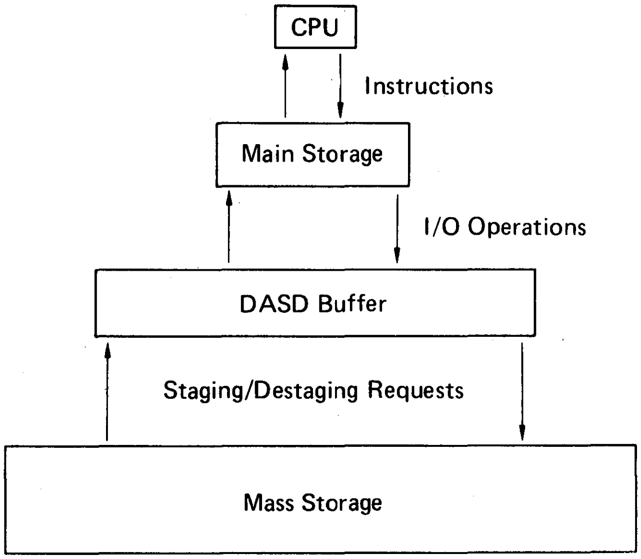


Figure 1—Hierarchical store - Data movement

devices attached to a given system on which storage volumes are mounted.

MSS resolves this limitation through a virtual device concept. Because DASD (called "staging buffers") is used in MSS as a buffer to the Mass Storage Facilities, the one volume per device relationship is no longer necessary. Only the active portion of a volume must be on DASD at a particular time; therefore, the remaining space of the DASD staging buffers can be used for portions of other active volumes. A virtual to real mapping capability is added to the Storage Controllers. Each Channel connection to a Storage Controller with MSS has the ability to address 64 unique volumes, independent of how many real devices are being used as staging buffers.

A virtual device approach makes more data addressable at any time without the cost of a corresponding number of I/O devices; this in turn allows larger data bases to appear to be on line, opening up new approaches to selected data base applications.

On real DASD volumes (non-staging), a user must put data on volumes that tend to be active at the same time, even if they are unrelated. Many problems can result, such as backup, security, etc. A virtual device concept eases the problem of space management on DASD volumes.

All space in MSS is managed in terms of DASD volumes. A pair of cartridges is assigned a name, and this name represents a specific DASD volume. This pair of cartridges is called a Mass Storage Volume.

To gain addressability to a Mass Storage Volume, the operating system (as a result of the active user programs) requests a specific Mass Storage Volume be mounted at a specific virtual I/O address. The mount command is directed to the Mass Storage Controller (MSC). The mount command allows MSS to translate an access to a specific Mass Storage Volume when I/O access is made against the virtual device. If the specific cylinder required by the CPU (1/404th of a Mass Storage Volume) is already on DASD, an I/O operation proceeds. If not, and data is being accessed, the MSC causes the cartridge containing the cylinder to be placed on a Data Recording Device (DRD), and the data contained in that cylinder to be transferred to the DASD staging buffer. The transfer path is to the Storage Controller, and from the Storage Controller to the Staging Buffer (DASD device) avoiding the extra load on channels and memory. The I/O operation then proceeds as before. If the Operating System knows which cylinders will be accessed, it can cause the MSC to stage only those cylinders containing the data set; reducing the number of times cartridges need to be accessed.

It should be apparent that the total amount of DASD Staging Buffer Space is related to the total size of the currently active data sets, and not to total number of active volumes. DASD is being used as a buffer to the MSF, and the actual device characteristics are architecturally unimportant.

CONFIGURING MSS

The objective of any configuration is to select the components required to do the work load of a specific customer. Excess performance capability, either as a result of the configuration or the design of the system, yields nothing useful to the customer and adds to the cost of the system. Systems must be configured to handle the highest work load case with the responsiveness required to meet work schedules or human factors requirements in interactive environments. Availability and security may also affect a configuration but are not the subject of this paper.

In evaluating a Storage System, some terms must be established to describe performance. The basic objective of storage is the transfer of data to or from a CPU's Channels. The total amount of data transferred is a measure of throughput (total work) while the time to access data becomes a measure of responsiveness.

Current systems can be measured using various techniques to determine the total amount of data transferred into or out of storage over given units of time. A basic difficulty in measuring a system is that, by definition, the measurement is based on past performance. Past performance provides an adequate measure only if the environment is static; which is seldom. A knowledge of existing applications growth and new application requirements must be acquired and factored into planned configurations.

In a storage hierarchy such as MSS, the Direct Access Storage must have sufficient performance capability to supply all accesses required by the CPU's main memory. Direct Access Storage is also involved when data is staged from the Mass Storage Facility or destaged to the MSF. It must have sufficient capacity to hold all open data sets and additional capacity to maximize the reuse potential of data. MSF must have transmission capability to handle the data being staged or destaged, and the ability to handle the cartridge movement involved. It must have capacity to contain all the data, backup, archival and active.

A work load analysis of the total system is required to choose the proper configuration.

The following is a hypothetical work load compiled to illustrate the demand that might be placed on the major interfaces of a storage hierarchy such as MSS. (See Figures 2-6.)

Figure 2 represents the I/O work load over time, and the sustained data rate across the CPU's channels, with tape and DASD averaged over one-hour periods. Each vertical bar reflects the combined effect of operator mount delays, operating system overhead, device raw data rate, channel/control unit configuration and demand for data by the programs that were active in that hour.

At any point, the Input/Output demand can be limited by the speed of the CPU and the number of I/O bytes of data required per instruction executed, or by the number of programs that may run concurrently. Peak throughput of the entire system is always limited by full utilization of the CPU's processing cycles. If the I/O is configured so it never limits the CPU to less than 100 percent utilization (and the cost of this I/O is not prohibitive) an ideal throughput system can result. Full utilization also requires an unending supply of work and response time which may be exceptionally long. Figure 2 is more typical, showing peaks and valleys in the I/O work performed by the system. Peaks represent 100 percent utilization and valleys represent the lack of work to do; or a system being limited by operations, number of available devices, or the mix of programs in the CPU.

The total amount of data staged and destaged is determined by the capacity of DASD and by how the data is being used. The potential for reuse of data can be understood by making an assumption about DASD capacity, and observing the resultant staging rate between DASD and MSF. First, assume there is only enough DASD capacity to contain the total size of concurrently active data sets; but with no additional capacity to retain data on DASD, then observe the demand on the MSF for data sets and data rate. Next, assume there is enough DASD capacity to contain the total size of concurrently active data sets, and there is additional capacity to retain data on DASD after its active use for longer and longer periods of time. Now, if a data set is reused within that retention period, it does not have to be staged again nor is it necessary to handle the cartridges. Figure 3 represents the results of such an analysis. The plot shows the reduction in average MSF data rate as the retention period on DASD is increased.

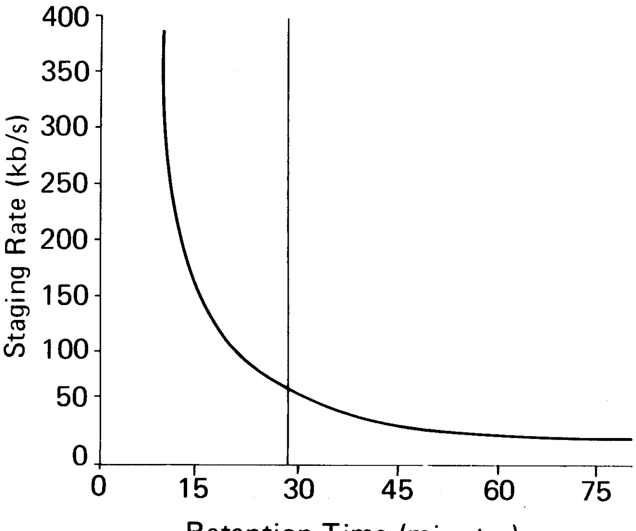


Figure 3—Staging data rate and DASD capacity relationship (72 hour example)

Once a retention period has been selected, it is more accurate to look at the required average capacity over smaller increments of time. Figure 4 shows the average capacity per hour that is required to contain all the active data sets during the time of their activity, plus the additional time included for optimum reuse. Given the average capacity to optimize the reuse of data, a closer analysis of

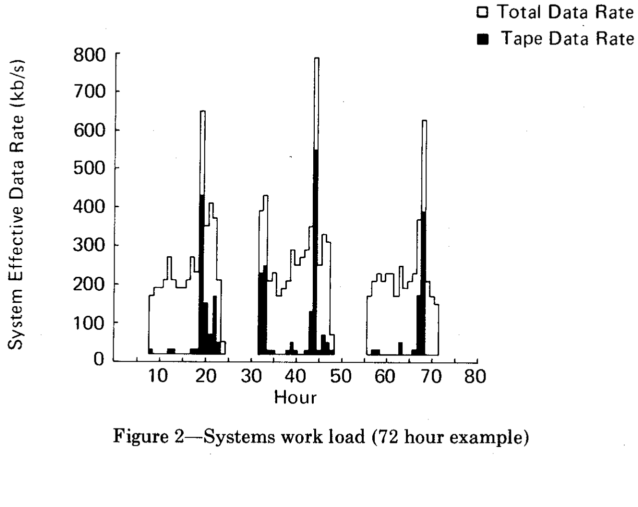


Figure 2—Systems work load (72 hour example)

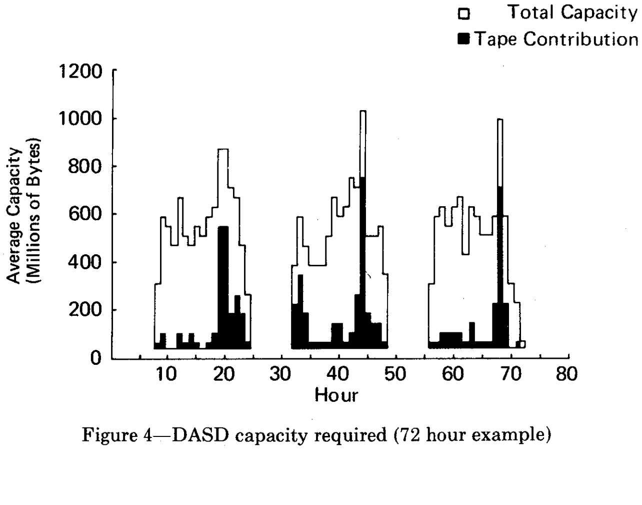


Figure 4—DASD capacity required (72 hour example)

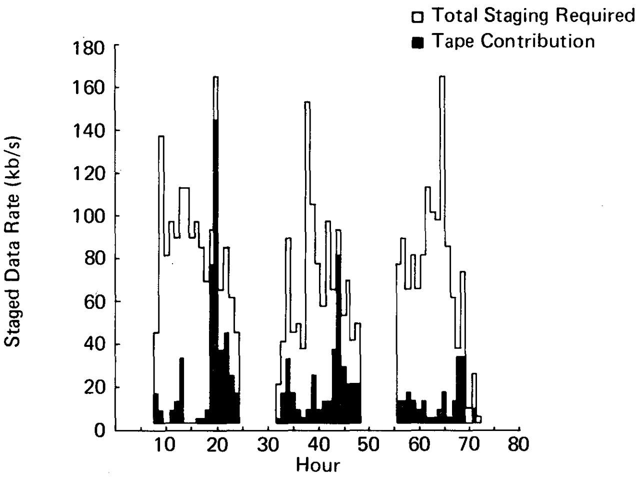


Figure 5—Data staging rate required (72 hour example)

MSF activity can be made. It must take two forms, the first is the total amount of data staged and destaged, the second is the total number of individual data sets handled. Both are necessary because they represent different work loads.

Figure 5 shows the staged data rate in each hour of the analysis. The Storage System must be configured so this demand can be met, and so it relates directly to the amount of DASD activity introduced by the storage hierarchy. The number of paths required for staging and destaging also result from this analysis.

The number of data sets being handled at the MSF level of the storage hierarchy can be used to determine the work load in terms of cartridge moves. Figure 6 shows this work load, stated in terms of cartridge moves per hour. The size of the data sets, and the Mass Storage Volume organization must be considered to translate data set accesses to cartridge moves.

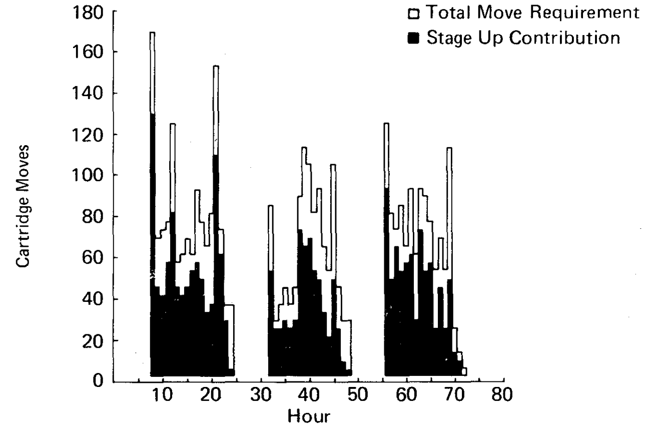


Figure 6—Access rate (72 hour example)

SYSTEMS CONTROL

The preceding sections presented an analysis of the work load relationship that could be expected on the major interfaces of a storage hierarchy. If the system is designed with work load relationships in mind, then maximum reuse potential must be achieved. In addition, the control system must function in large establishments which typically have multiple CPU systems controlled by separate operating systems. It is necessary to have, therefore, a common point of intelligence to control both the concurrent use of data and the staging buffer replacement algorithms.

The MSS has such a control system, called the Mass Storage Control or MSC. In MSS, regardless of how many Mass Storage Facilites exist, one and only one MSC controls the hierarchy. Some MSS models have two MSCs; however, the second serves only to improve availability. The MSC cannot perform the entire task without also having control over allocation of volumes to devices. Because of this, MSS is designed to allow the MSC to control the functions of space management, while the control of data and allocation is within the operating system. An interface between MSC and the operating system is a necessity.

In IBM System 370 architecture, a program gains access to data by having a volume mounted on an addressable device, as discussed previously. In a single CPU environment, the operating system is always aware of which volumes are currently mounted. In a multiple operating system environment, such as illustrated in Figure 7, the systems can take actions independent of each other. In addition, a storage hierarchy such as MSS is attempting to maximize the reuse of data. If any residual data remains on the staging buffer and the same using system or another system reuses that data, the MSC must know where that data resides to prevent unnecessary staging. Another condition which may exist is where one CPU is currently using a data set on a specific volume, and another CPU needs access to the same data set, or another data set also residing on that volume.

The virtual device concept provides a convenient solu-

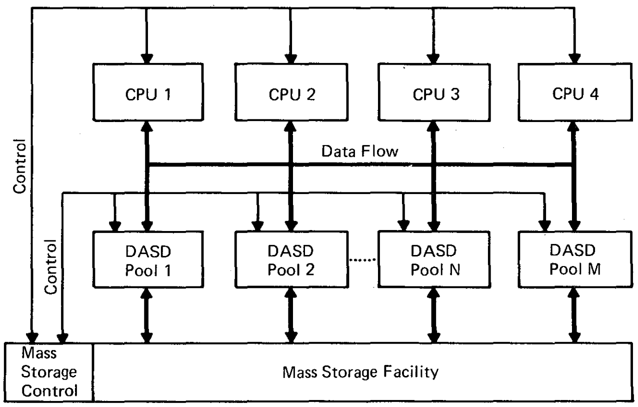


Figure 7—Multiple systems environment

tion to the following requirements. First, assume that CPU 1 has access to a volume and it has allocated that volume to DASD POOL 1. Now, assume CPU 2 wants access to another data set on that same volume. If the operating system in CPU 2 were to independently allocate that volume, it would be pure chance if it selected a device address associated with POOL 1. In MSS, the CPU that is in the process of allocation, uses the MSC control interface to learn if the volume in question is currently mounted, or if any residual data from a previous usage of the volume exists anywhere on the staging buffers. If either condition exists, the operating system retrieves the information from the MSC and chooses a virtual address associated with the DASD POOL that contains the data. The MSC then conditions the Storage Controllers associated with that DASD POOL to access to the specific unit address maps to the desired data. This makes it possible for CPU 1 to access a specific volume on one virtual device address while at the same time, CPU 2 has access to the volume on a different virtual address. The converse is also true, CPU 1 and CPU 2 can be accessing different volumes on the same virtual address.

The above described control makes it possible for multiple CPU's, which may be controlled by different operating systems (OS/VS1 and OS/VS2 for example), to concurrently share the DASD buffer space without having a common allocation system. As many as six different virtual addresses can be mapped to the same volume, or each may have different volumes associated with them. Each Storage Controller used in MSS can have up to three channel interfaces, and one or two Storage Controllers may access the same DASD POOL. Each interface can have up to 64 virtual addresses, independent of the actual number of DASD devices in the pool. This means that 64 volumes could be mounted in that pool with six alternate paths to each, or that up to 384 volumes could be mounted with no alternate paths.

To control sharing within MSS, two mechanisms are used. A logical device reserve/release function is provided and is functionally compatible on virtual devices with current IBM 3330 devices. A second mechanism is provided called exclusive Mass Storage Volumes. This function allows the volume to be defined to MSS so the MSC can allow only one CPU access at a time. Actually, if the data on the volume is read only, or only one user is in the update or write mode, there is usually no reason to prevent concurrent access.

SPACE MANAGEMENT

Managing space on DASD volumes is considerably more complex than managing a tape library. This is due to the capacity of DASD volumes and the function they provide. Typically tape volumes contain data from a single data set, while DASD volumes contain data from many data sets. If the data sets have different life times (expiration dates), and are of a variety of sizes, the result can be fragmentation of the space. This is further complicated by different security and backup requirements. It, therefore, is desirable to provide new functions to ease the task of managing space within MSS because of its capacity (equivalent to up to 4,720 Packs of a 3336, Model 1).

Again, it is necessary to divide the control between MSS and the attached CPU operating systems. The division is made to manage space within the MSC, and to manage the content of that space with the existing catalogue functions of the operating systems; also to complement the catalogue functions with new software functions called Mass Storage Volume Control (MSVC).

The objectives of MSVC are to aid the users of MSS in space allocation and space reclamation and to provide a management reporting function so that use of space within MSS can be monitored. The main parameters of space management are accounting, data set size, life cycles, security and backup. Because these parameters vary widely between installations, a framework is implemented to give customers control with as little implementation effort as possible.

MSVC maintains an inventory of Mass Storage Volumes. This inventory can be structurable into groups of volumes. The MSVC maintains information about ownership, space usage, space allocation defaults, backup volumes, retention dates, and location of the Mass Storage Volume if it is removed from the system. Volumes can be grouped by data set sizes, user group, security or any other parameter the customer chooses. Once a structure is chosen, attributes can be applied to the entire group or to individual volumes within the group. It should be possible in most cases to use MSVC so that the space management within MSS is automatic. MSVC also uses the control interface to MSC to manipulate the volumes physically or logically. For example, volumes may be removed from MSS for security purposes, or may be made inaccessible within the system. Volumes may be copied within MSS without data movement through the CPU's memory or channels, and the copy is inventoried by MSVC, and access to the copy is controlled by the MSC.

CONCLUSION

As stated earlier, the number of times and the amount of this time, a program or task has to wait for data to appear on DASD or has to wait for space on DASD to store new data, is a measure of the visible impact of the storage hierarchy and the overhead it introduces into the I/O system. The effect on total system performance can only be understood by comparing the projected overhead with the existing overhead.

Again, averages are used to demonstrate the potential for improvement. Figure 8 is a comparison showing delay times in a half-inch tape environment where tape volumes are handled by operators; versus the delay times caused by MSS. Figure 8 assumes MSS has been configured to the optimum reuse DASD capacity and has a sufficient number of access paths to handle the peak work loads demanded by the job mix. It also assumes the events of data

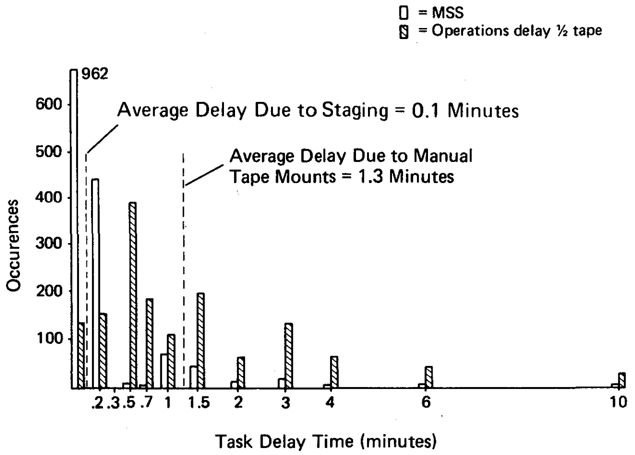


Figure 8—Estimated task delay time (comparison example)

set usage are essentially the same in both environments. The potential for improvement in this specific case is approximately one order of magnitude.

The preceding analyses allow the following observations to be made relative to the MSS storage hierarchy.

1. The best possible performance of the hierarchy is equal to that of the devices used in the top level of storage. For MSS, it is the DASD level.
2. The total cost of the Storage System is dependent on the size of data sets, their reuse characteristics, and the amount of data that must be transmitted to and from the CPU's main memory.
3. Mass Storage can be designed with considerably less data transmission capability than devices which must respond directly to the needs of an active program.
4. The measurable impact of the hierarchy is the length of time it takes to make data or space available to the CPU channels. The total amount of storage space at the DASD level of the hierarchy can have a greater effect on this response time than the transmission rates between the levels of the Mass Storage System.
5. A virtual device architecture allows the storage hierarchy to adapt new technologies without changing or impacting the user's interface to data. The hierarchy can be expanded to include additional levels of storage within this basic approach.

RECOMMENDATIONS

Understanding the performance of a storage hierarchy is difficult because of its wide range of applications. It is relatively clear that properly configured, a storage hierarchy performance will be comparable to manually mounted tape/disks in local and remote batch applications where data set sizes are within the 50 $\times$ 10^6 byte range and where reuse characteristics are present. Storage hierarchies are particularly interesting in remote compute applications. The use of storage hierarchies in interactive data base applications, however, needs considerably more understanding in most cases.

There are a number of applications which require very large data bases but have a low rate of access. If this rate of access is less than delivery capability of cartridges in MSS, then the analysis is simple. If, however, the total work load on the storage system includes local and remote batch, interactive compute, interactive data base and utility type functions (backup, interchange, archive, etc.) understanding the total system is very complex.

Usually this type of complexity can be understood by modeling. In the case of a storage system, however, proven modeling techniques are not available. Analytical models require that input parameters are some sort of distributions or averages. Simulation models require large quantities of trace data in order to cause sufficient activity at the lowest level of the storage hierarchy. The availability of trace data assumes the application has been done somewhere. Data organization must be included to relate data accesses to MSF accesses. Also, clustering and sequencing of requests must be related to data organization and understood relative to retention times in the higher levels of storage.

The recommendation is to use caution in approaching data base applications where there is a dependency on "80 percent" of the activity versus "20 percent" of the data (or some other rule). Modeling techniques must be developed which allow total systems to be analyzed. Data must be collected and understood which can indicate the predictability of when data will be used and how it relates to other data. Lastly, it must be recognized that these applications will coexist within the same complex; therefore, the analysis must encompass the old as well as the more exciting, new applications.

Charge-coupled devices for memory applications

by GILBERT F. AMELIO
Fairchild Research and Development
Palo Alto, California

INTRODUCTION

Since the invention of charge-coupled devices (CCD) in late 1969,[1-3] the potential of the charge-coupling concept for digital memory has been recognized. Applications to image sensing and signal processing requirements have, however, preceded the application of CCD to memory. This occurred because relatively simple charge-coupled devices offered new performance potentials for image sensing and signal processing. Nonetheless, the basic shift-register nature of CCD implies that its greatest opportunity for widespread application lies in the highly competitive area of high density semiconductor memory. Today, sophisticated CCD memory components are emerging.

Although a CCD memory can be configured in a number of ways, all are basically serial in nature and, hence are block-access oriented rather than bit-access oriented. The charge-coupled device is characterized by high packing density, low power dissipation, and a structural elegance which will lead to very low cost. Besides being a natural semiconductor replacement for rotating drums, discs and other peripheral memories, CCD will ultimately find application in many main-frame requirements as systems architecture evolves to take advantage of these fast, lower cost block-oriented components.

This paper begins by describing the general properties of CCD memories, focusing on speed, power, temperature characteristics, and interface requirements. Next described are three basic classes of CCD memory components—the Synchronous, the Series-Parallel-Series, and the Line Addressable. As a specific example, a Synchronous configuration with 9216 bits and fabricated with Isoplanar, buried channel processing is described in detail.

GENERAL PROPERTIES OF CCD MEMORIES

Speed

The movement of charge from one CCD electrode to the next is inherently fast, ultimately limited only by the carrier saturation velocity. Registers which shift data efficiently at frequencies in excess of 100MHz have been reported by Esser.[4] Improvement by factors of 2 or 3 over these values is only a matter of time. Such speeds will not be as easily achieved in useful CCD memory components. In a practical memory component, significant peripheral and interface circuits are required. These include read/write logic, level converters, sense amplifiers, and I/O buffers. In current practice these circuits are implemented, as is the CCD, by N-channel MOS technology. For standard voltages and present design rules, such circuits are limited to about 10MHz with 5 to 7MHz representing a more practical current upper limit for production devices.

Power dissipation

The dynamic, non-equilibrium operating mode of the CCD element leads to an almost ideal energy transfer condition where on-chip power dissipation is associated primarily with the movement of signal charge (data). If, on the average, one-half the bit sites contain charge, the speed-power product for typical parameters is 0.2*pJ* per bit transferred. As an illustration, consider a simple series shift register block of N bits. The average power dissipated on-chip for a data frequency of f_c is given by:

$$P=2N(0.2\times10^{-12})f_c. \tag{1}$$

If $N=1024$ and $f_c=5\text{MHz}$, then

$$P=2mW \tag{2}$$

which is a small power for a memory of this size. Moreover, in other configurations where most of the data is moving slowly or not at all, the power per bit is averaged downward very much further.

A much more significant term is the power dissipated in the drivers to the CCD register. Since the CCD electrodes present a substantial capacitive load, the drivers dissipate power according to the general law for reactive power,

$$P=CV^2f_c$$

For the simple series shift register considered above, the capacitance per bit for an Iosplanar, buried channel structure is approximately $C_b=60fF$. For $V=10V$ and

$f_c=5$MHz, the driver power becomes

$$P=2NC_bV^2f_c=60mW \qquad (3)$$

Although this is much larger than the real power dissipated in the CCD, it is quite acceptable. The driver power requirement can be averaged downward by slowing the clock frequency when the memory is not actively used.

Perhaps the largest on-chip power dissipation is in the peripheral circuitry. Although it is difficult to be quantitative without a particular memory configuration in mind, a power dissipation of one to two orders of magnitude higher than the CCD storage elements can be expected.

Temperature behavior

The CCD storage element is dynamic and therefore, must be periodically refreshed similarly to a dynamic MOS RAM. The frequency of refresh is a sensitive function of temperature. At room temperature (25°C), the element storage time is of the order of one second. This value decreases by a factor of two for every 9°C increase in temperature up to approximately 70°C. Above this it decreases even more rapidly, until at 125°C, it is falling a factor of two every 4.5°C. For an ambient temperature of 70°C, a typical die temperature might be around 90°C. Since under these conditions the storage time is approximately 5ms, the memory must be refreshed more frequently than at lower temperatures. This increase in refresh rate increases power dissipation which, in turn, increases the difference between the ambient temperature and the die temperature thereby further increasing the required refresh rate. The power dissipation, therefore, increases in a power law fashion as the system ambient temperature rises. Conversely, at artificially lowered temperatures (<25°C) the power and time required for refresh rapidly becomes insignificant. For example, at −30°C ambient, the storage time is approximately one-third of a minute which thereby renders the memory close to non-volatile from a power dissipation standpoint. Future memory systems using CCD may find it advantageous in many instances to incorporate cooling for increased system performance.

Interfacing and packaging

As a result of the extensive use of NMOS technology for the on-chip peripheral circuits, CCD memory components should be no more difficult to use than dynamic MOS circuits. Indeed, dependent on design, the timing and other requirements may be superior to MOS.

The similarity of CCD memory to MOS memory will lead to the use of essentially identical packaging technology, which includes side-brazed ceramic, cerdip and plastic. The designing and qualification cycles should be rapid.

MEMORY CONFIGURATIONS

The inherent digital CCD configuration is that of a shift register with a serial-in/serial out operation where all bits are simultaneously shifted. CCD memory elements can be formed by using these shifts registers directly or by combining the shift registers with refresh, sense and decoding circuitry to construct more sophisticated memory chips. There are three basic organizations for memory applications:

(1) Synchronous

The synchronous organization is one in which all shift register segments are clocked simultaneously. For example, the well-known serpentine configuration is synchronous. This organization (Figure 1a) is obtained by connecting in tandem several sequential multi-stage shift registers by means of refresh cells. Internally, all data bits are simultaneously shifted through all cells of the register and characteristic of the synchronous organizations, the internal shift frequency is equal to the input/output data rate. An important advantage of the serpentine configuration is that it permits the construction of extremely long shift registers with excellent low frequency response even at elevated temperatures where leakage currents become significant. A disadvantage of this organization is that power dissipation is higher at a given operating frequency than the other approaches because all bits of data are moving at the same frequency. Additionally, the clock power requirements are high for a large capacity memory since the clock-line capacitance is relatively high. Several serpentine shift registers can be combined on a single chip using common control signals and an address decoder which enables selection of one or more of the serpentine registers. Because all of the serpentine memory operates at the same frequency, it can operate over a wide range of frequencies, which is particularly useful for data buffering applications.

(2) Serial-Parallel-Serial (SPS)

The power efficiency and effective packing density of the basic CCD shift register can be increased by performing a serial-parallel-serial (SPS) manipulation of the data. An SPS memory arrangement is shown schematically in Figure 1b. The input rate is determined by the clock frequency of the horizontal registers. After filling the top horizontal register with data, a parallel operation is performed in which all of this information is shifted into the vertical registers. Thus, the vertical registers run at a frequency equal to N^{-1} of that of the horizontal registers where N is the number of bits in the horizontal register. If it is assumed that the total number of bits is $N\times M$, the delay from input to output is NM/f_h where f_h is the horizontal clock frequency. However, the number of stages of shift register through which any one bit is transferred equals only $N+M$.

One advantage of this configuration is that the power dissipation is low because most of the data bits are moving at the slow (vertical) clock rate. This reduction in power dissipation is obtained at the expense of an additional set

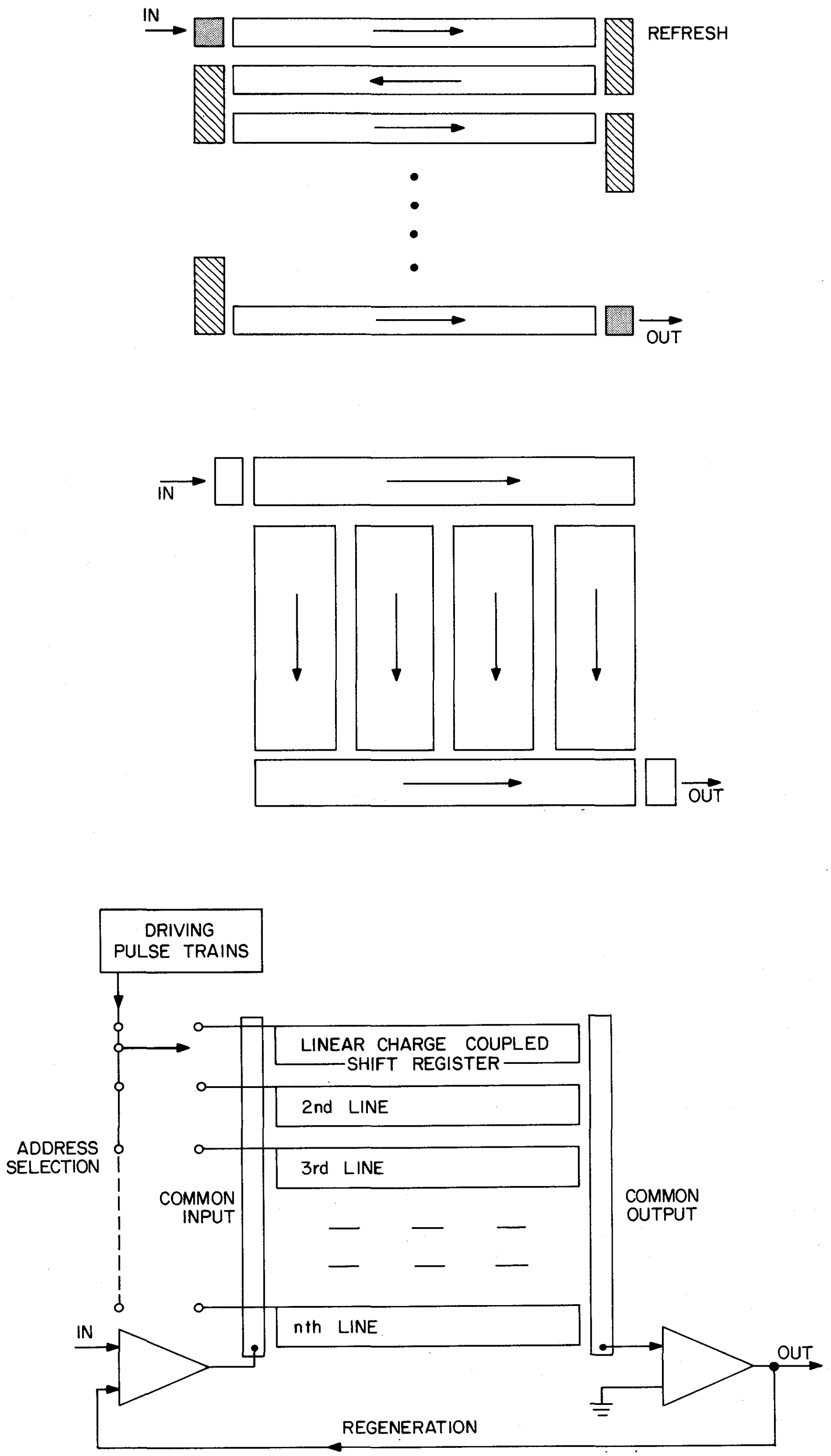


Figure 1—A schematic representation of a charge-coupled. (a) serpentine shift register organization. (b) serial-parallel-serial (SPS) shift register organization. (c) line addressable random access memory (LARAM) organization

TABLE I

Memory Class	Application	CCD Memory Order of Preference
FAST ACCESS MEMORY	Fast Access Bulk Storage	LARAM
	Main Memory Mini CPU	Synchronous
	Signal Processing	SPS
	Multistation Data Collection	
	Disc Enhancement	
SERIAL MEMORY	Airborne Mag Tape Repl.	
	Commercial Disc	SPS
	Commercial Tape	LARAM
	Lightweight Field Data Collection	Synchronous
	Communications Buffers, Large Record	
BUFFER MEMORY	Multiplex Communications Buffers	Synchronous
	Buffered I/O controllers	SPS
		LARAM

of clocks for the vertical registers. Another advantage is flexibility in timing if the vertical clocks are programmable. Additional flexibility can be achieved with separate input and output clocking.

A disadvantage is the long delay between input and output. This limits the low frequency and high temperature operation of the memory, because of leakage currents. Similarly, the access times for SPS memory chips are generally longer than those for other organizations. It should be noted, however, that the data rate can still be very high. Overall, the SPS characteristics make it very attractive as a potential replacement for mechanical memories.

(3) Line-Addressable

Although a true random bit-access capability is inherently precluded by the very nature of the CCD storage structure, a novel organization has been conceived which does provide a pseudo-random access with access times in the tens of microseconds. This is the line-addressable random access memory (LARAM) configuration which is an integration of CCD and MOS memory concepts. One form of the LARAM is illustrated in a simplified schematic form in Figure 1c. Basically, the memory is composed of an MOS address selection matrix and a number of CCD sequential shift registers, where each register represents a line. Selection of an address causes the driving waveforms to be applied to the chosen line (register) to initiate read-out/write-in/ or refresh or information in that register.

This configuration allows an access time that is essentially dependent on the number of elements per line. In addition, since only one line is operative at any one time, clock capacitance and power dissipation are minimal. Dependent on the stack configuration, a memory system using a line-addressable structure can be either word-organized, where each line represents one or more words, or bit-organized, where each line contains one particular bit of a number of words. Since the data is, in general, not moving most of the time, this type of device organization has the most stringent requirement on dark current uniformity. Nonetheless, this organization is the most flexible and will find greatest application in cache buffers, swapping stores and mainframes.

Other organizations which are basically derivatives of those above include among others, the multiplexed electrode-per-bit[5] configuration, the interlaced SPS[6] and the addressed drum type structure.[7]

In the multiplexed electrode per bit configuration each *electrode* is used as a storage site. The clocking is such that only 1 of N electrodes is clocked at a time thus propagating a "hole" through the stored data. For the price of an N-phase clock ($N \gg 1$) the packing density of the memory can be doubled. The interlaced SPS is just like any other SPS except there is one horizontal electrode rather than one bit for each vertical register. Clocking is such that the odd and even vertical registers are alternately filled from the odd and even electrodes of the horizontal input register. At the output, this sequence is pursued in an analogous but reverse manner. The addressed drum type structure is a synchronous organization similar to the serpentine configuration except each loop closes back on itself. Access to these independent loops is by means of an address decode for the I/O.

Several organizations will emerge as standard CCD memory products as a result of the complex trade-offs between system requirements and device economies. A qualitative comparison of the three basic organizations for different applications is shown in Table I. Note that in the critical area of fast access memories (FAM's), the LARAM is the preferred configuration. This choice derives from the fast access time and low clock capacitance characteristic of this organization which permits easy memory expansion. In addition, the power dissipation is low and the organization leads naturally to a high density layout. As a result of these unique features, the LARAM will play a major role in the application of CCD to system design and layout. For serial memory, the SPS is preferred because of its simplicity. Finally, for buffer memory synchronous architecture is preferred because of its versatility and ease of use.

EXPERIMENTAL RESULTS

In order to put the above remarks in a realistic frame of reference, this section deals with a specific CCD/NMOS

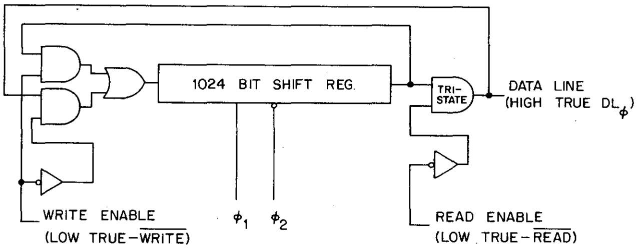

1024 BIT SHIFT REG.
TRI-STATE
DATA LINE (HIGH TRUE DL$_\phi$)
WRITE ENABLE (LOW TRUE-$\overline{WRITE}$)
ϕ_1
ϕ_2
READ ENABLE (LOW TRUE-$\overline{READ}$)

Figure 2—A logic diagram for one of the nine channels in the CCD450

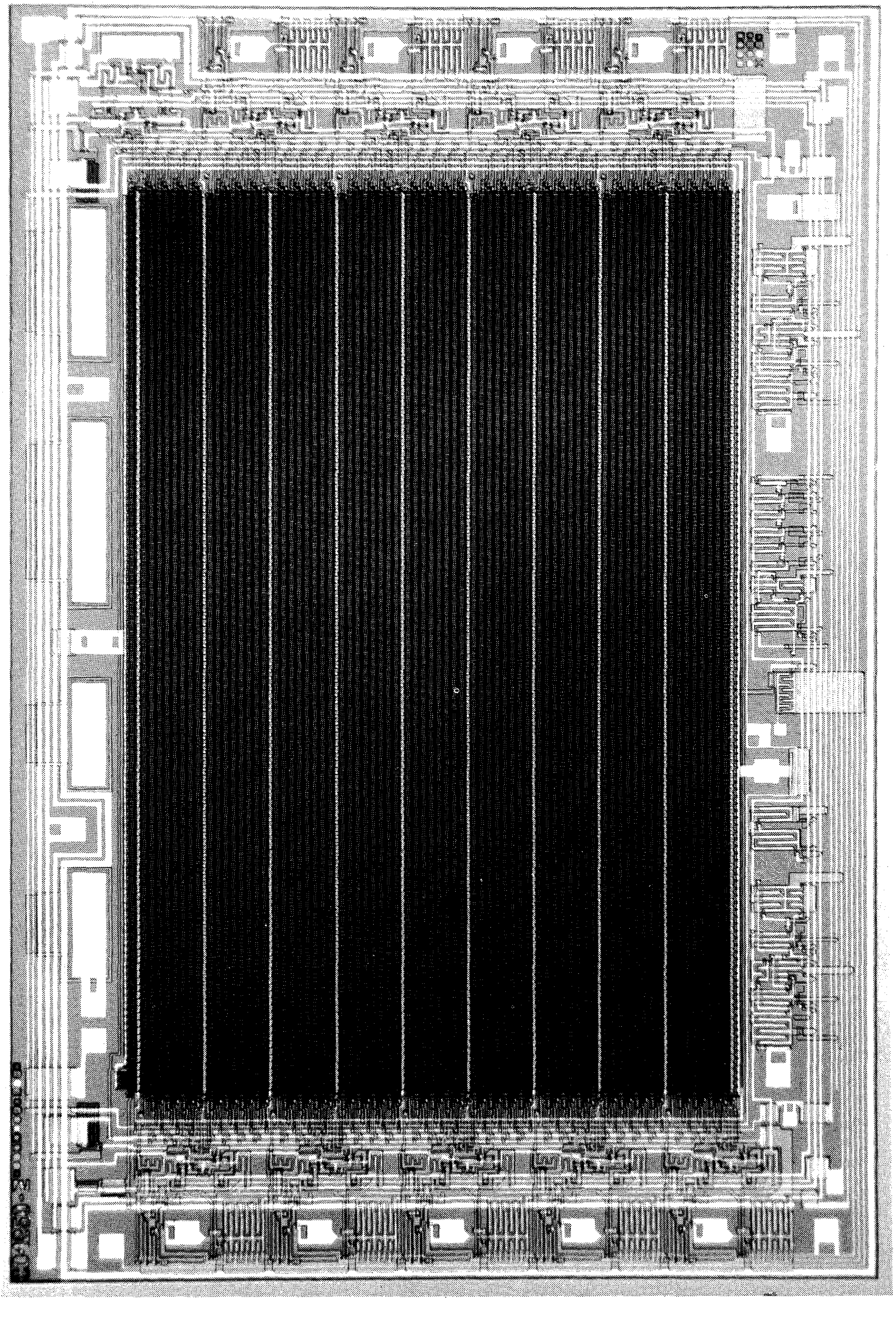
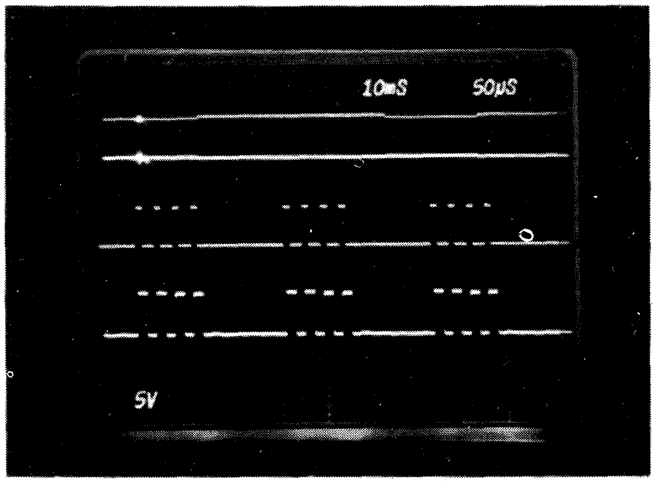
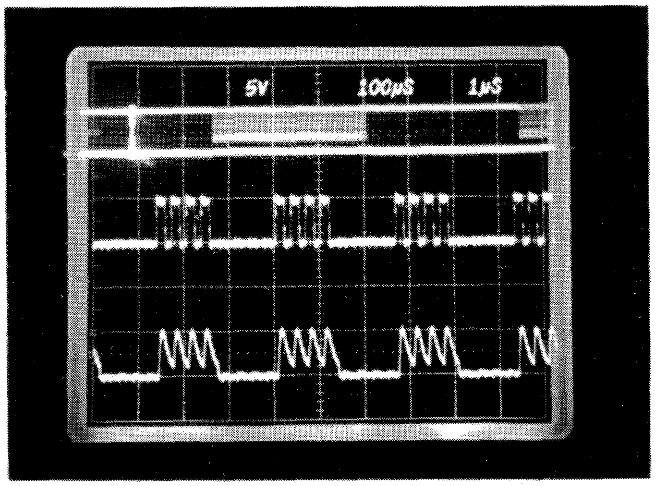


Figure 3—Photomicrograph of the CCD450



(a) 50 KHz Data Rate



(b) 3MHz Data Rate

Figure 4—Oscilloscope traces of one channel of CCD450 operating at 50KHz and at 3MHz. The lower two traces show input data and output data respectively

memory component which has been developed at Fairchild. This part is known as the CCD450.

Description

The CCD450 is a serial storage memory consisting of 9216 bits which are organized into a format of 1024 bytes by 9 bits. This architecture is realized by the use of nine shift registers each containing 1024 bits. Since these registers are shifted in parallel, nine-bit bytes are stored or retrieved in a byte-serial mode.

The component represents a significant advance in semiconductor memory. Although it contains all the logic necessary to facilitate ease of use the average density is less than 3 mil^2/bit. The fabrication of the memory is little more complicated than comparable *N*-channel MOS circuits with only one additional photomasking operation. On-chip TTL-to-MOS level conversion permits all logic and data lines to be TTL compatible. On-chip timing reduces the system requirements to simple two phase clocks. Bi-directional tri-state data lines and commonality of logic for I/O control permit the memory to be packaged in a 0.3 inch 18 pin standard package.

A logic diagram for one of the nine registers is shown in Figure 2. Each register is accessed by its own bi-directional data line, but all nine registers are serviced by common data-transfer clocks (ϕ_1 and ϕ_2) and control functions $\overline{\text{READ}}$ and $\overline{\text{WRITE}}$. Not shown on the logic diagram, but also common, are the DC power supply lines and a data-enable (DE) line. The data lines are driven with a tri-state buffer, thus providing a "wired OR" capability at the output. This eases memory system expansion, eliminating the need for interfacing components on the PC board.

The CCD registers are organized in a serpentine fashion with refresh turn-around-cells every 128 bits. The CCD structure used is a buried channel, gapless structure with ion-implanted barriers. The MOS circuits used to provide timing, charge detection and level conversion are fabricated using Fairchild's n-channel silicon gate Isoplanar$^{(R)}$ process. Figure 3 is a photograph of the chip.

Performance

The operating modes of the circuit include: read, write, read/modify/write, and recirculate. Furthermore, depending upon the data transfer rate, the recirculate mode can be broken down into a search mode and a standby mode.

The basic timing is established by the two clocks (ϕ_1 and ϕ_2). During ϕ high time, the logic is reset, data is shifted by ½ bit and the mode control level conversion from TTL to MOS levels is accomplished. During ϕ_2 high time the data is shifted by ½ bit, output charge (or lack of charge) is sensed and presented to the output and, simultaneously, is written into the first CCD stage. The charge level written into the first CCD well is controlled either by the output charge level or the data pin state depending upon whether the selected mode is *recirculate* or *write*. Figure 4 shows one channel of the memory operating in a write/recirculate/read mode at 50KHz and at 3 MHz.

TABLE II—Summary of CCD450 Characteristics

ORGANIZATION:	1024 BITS x 9 BITS
OPERATING FREQUENCY:	3 MHz MAX READ, WRITE OR RECIRCULATE 2 MHz MAX READ/MODIFY/WRITE 50 KHz MIN STANDBY
INPUT/OUTPUT INTERFACE:	TTL COMPATIBLE BI-DIRECTIONAL DATA BUS
READ/WRITE LOGIC INTERFACE	TTL COMPATIBLE "LOW-TRUE" LOGIC INPUTS
CCD CLOCKS:	TWO-PHASE CLOCKS (ϕ_1 AND ϕ_2)
DATA ENABLE CONTROL:	DISABLES READ/WRITE BUFFER FOR LOW POWER STANDBY OPERATION (0-12 VOLTS)
POWER DISSIPATION:	READ-AT 3 MHz-250 mW TYP. RECIRCULATE AT 3 MHz-50 mW TYP. STANDBY AT 50 KHz-30 mW TYP.
CLOCK CAPACITANCE:	400 pF MAX.
VOLTAGE REQUIREMENTS	ϕ_1, ϕ_2 AND DE 0-12V CLOCKS V_{DD} +12 Vdc V_{CC} +5 Vdc
PACKAGE:	18-LEAD D.I.P. (300 Mil WIDTH)

During the recirculate mode, only those circuits required for internal data transfer need to be active. This selection is accomplished by bringing the data enable (DE) line to its low state, thus disabling the read and write logic. While in this state, the chip will disregard the $\overline{\text{READ}}$ and $\overline{\text{WRITE}}$ lines, thus removing all constraints on these lines during the recirculate mode. This mode is thus useful for either a high speed "search" operation, or a low speed, low power standby operation.

A low power standby mode can be obtained by reducing the operating frequency. Further power reductions can be obtained by reducing the duty cycle, operating with both ϕ_1 and ϕ_2 low for much of the cycle. Table II lists the device characteristics.

Application

The CCD450 is most conveniently envisioned as a byte-organized dynamic shift register with sufficient overhead functions to ease its use considerably. As such, it is most attractive for terminal applications where the byte organization and low power are highly desirable. In this application, it replaces more than nine packages, saves several square inches of board space and reduces power dissipation by more than an order of magnitude. The low power recirculate mode permits battery back-up for non-volatility and portability.

CONCLUSIONS

The charge-coupled device will play a major role in future semiconductor memory systems. Its characteristics of low complexity, low power dissipation, high data rate and ease of use will result in rapid growth in numerous memory applications and produce concomitant reduction in cost/bit.

REFERENCES

1. Boyle, W. S. and G. E. Smith, "*Charge Coupled Semiconductor Devices,*" *Bell System Tech. Journal,* Briefs, 49, p. 587, April 1970.
2. Amelio, G. F., M. F. Tompsett and G. E. Smith, "Experimental Verification of the Charge Coupled Device Concept," *Bell System Tech. Journal*, Briefs, 49, p. 593, April 1970.
3. Tompsett, M. F., G. F. Amelio and G. E. Smith, "Charge Coupled 8-Bit Shift Register," *Appl. Physics Letters*, 17, p. 111, August 1970.
4. Esser, L. J. M., "The Peristaltic Charge Coupled Device for High Speed Charge Transfer," *1974 IEEE Solid-State Circuit Conference*, p. 28.
5. Collins, D. R., J. B. Barton, D. C. Buss, A. R. Kinetz, S. E. Schroeder, "CCD Memory Options," *1973 IEEE Solid-State Circuits Conference*, p. 136.
6. Erb, D. M., and C. V. Agnew, "A High Density Serial/Parallel/Serial CCD Memory with Interlaced Columns," *Session 3-B, Dev. Res. Conf.,* Univ. of Ca., Santa Barbara, December 1974.
7. Rosenbaum, Stanley D., and T. Terry Caves, "CCD Memory Arrays with Fast Access by On-Chip Decoding," *1974 IEEE Solid-State Circuits Conference*, p. 210.

Bubble domain memory systems

by JOHN E. YPMA
Rockwell International Corporation
Anaheim, California

INTRODUCTION

This paper presents bubble memory technology from a systems viewpoint. A perspective is given listing the characteristics of bubble domain memory relative to other available and oncoming memory technology. The application areas most conformed to bubble capability are identified. Bubble domain chip topologies and bubble memory systems organized to fit the application are discussed. Examples of actual devices being built are presented. Prototype systems are discussed, indicating their organization and interface command and timing structure.

BACKGROUND

The effort presently being put forth in the development of bubble domain memories is based upon three factors. First, the basic characteristics associated with bubble domain memory are attractive. Secondly, there are at least three places in the spectrum of memory approaches where bubble memories can fill vacancies in an effective manner. Thirdly, the technical problems associated with bringing bubble technology into fruition as hardware are being surmounted. By November, 1974 at least three companies had constructed prototype bubble memory systems which exhibited all the necessary functions required for a usable memory.

Those characteristics which make bubbles attractive for storage systems are several in number. Bubble memory is non-volatile. Its non-volatility provides data protection during power failures. More important, it allows the memory to be powered down when no data is being transferred. During operation, power consumption remains low with 10 Watts for a 10^7 bit memory as a reasonable figure. These power levels coupled with bubble memory bit density in the range of 2.5×10^6 bits per square inch allow compact packaging to be achieved without heat problems. Lack of mechanical motion and the periodic servicing associated with moving devices makes bubble memory a candidate wherever servicing is difficult or overly costly. Finally, bubbles extend the range of non-volatile, non-mechanical magnetic storage to sizes heretofore unavailable.

Tables I and II and Figure 1 show in greater detail where bubble memories fit with respect to other memory approaches. Three application areas where bubbles can provide added capabilities can be seen.

The first is an endless loop recorder with a capacity of about 10^6 bits. This application, at the small end of the capacity range can be considered as a non-mechanical alternative to the cassette or floppy disk. This type of device would be used for data loggers, text editors, point of sale terminals, and some portable applications where extremely low power and physical ruggedness is required.

The second application area is as a large block organized store with 0.5 to 1 millisecond access time at 0.050 cents per bit cost ceiling. This application fills the classical access time gap between 1 microsecond core memory and 8 millisecond head-per-track disc memory. Here bubbles offer a substantial improvement in block and access time for data processing application. In addition to commercial EDP usage, the non-mechanical nature of bubble memory is especially applicable for military and airborne applications.

The third application area is a large non-mechanical storage system for tape recorder replacement. The most common usage for this device is in spacecraft tape recorder replacements for satellite service. The next section describes a small model which has been built.

SERIAL RECORDER—A DESIGN REDUCED TO PRACTICE

Figure 2 is a block diagram of a serial recorder feasibility model developed for NASA by Rockwell International. The purpose of this very small recorder was to test and exhibit the many system features expected of bubble memory technology. A short listing of its characteristics is given in Table III. Effectively, the unit was comprised of three completely independent 20,000 bit tracks housed in a single module and synchronized to a single clock. Each of these tracks was free to run independently of the others. In this organization it was entirely reasonable for one track to be recording data at one speed while another was replaying data at another speed for processing.

Several significant operational features were established in this model. One important attribute desired by NASA was that of very low power. To accommodate this need, the recorder was designed so that each functional section was power strobed in accordance with its usage. The read circuit is energized only when reading; the write only dur-

TABLE I—Comparison of Memory Attributes

TYPE	VOLATILE	MECH	DATA ACCESS	SIZE BITS	COMM. COST/BIT
TTL RAM	Y	N	RANDOM WORD	10^3-5.10^5	5¢
MOS RAM	Y	N	RANDOM WORD	4.10^3-10^6	1.25¢
CORE	N	N	RANDOM WORD	10^5-5.10^7	0.7¢
BUBBLE DOMAIN	N (NDRO)	N	RANDOM BLOCK	5.10^5-2.10^8	0.03-0.05¢
FLOPPY DISK—DRUM	|	Y	SERIAL BLOCK	5.10^5-5.10^6	0.05¢
CASSETTE	|	Y	SERIAL BLOCK	10^6-10^7	0.04¢
HEAD/TRACK DISK—	|	Y	SERIAL BLOCK	10^7-2.10^8	0.08¢
MOVING HEAD DISK	↓	Y	SERIAL BLOCK	5.10^7-5.10^9	0.0025¢
TAPE	N (NDRO)	Y	SERIAL BLOCK	10^8-10^{10}	0.0001¢

ing write; and the entire recorder is turned off between operations. Successful performance of the unit under this mode of operation established several bench marks. Nonvolatility was established by the repeated rapid turning on and off of the system. More significantly, however, the ability to run for just a few cycles and perform all of the recorder functions reliably was established. In the byte read mode successive groups of 8 bits of data were read from the recorder in 8 cycle bursts. The ability of the bubble memory to be read immediately after starting without gaps was established. This ability to start and stop reliably was also used to make the recorder exhibit a pseudo variable speed characteristic, where data could either be recorded or played back at any rate from dc to the maximum operating frequency of 150 kHz. This was done by filling an input buffer at system rate, while emptying it with 150 kHz bursts of operation. This made the recorder synchronous to the system which it served. Finally, the power sequence circuitry developed established the feasibility of zero power standby modes in portable data logging applications where the amount of primary power available is at a minimum.

ENDLESS LOOP RECORDER

A version of a recorder more applicable to data logging is shown in Figure 3. This design utilizes 8, 100,000 chips to form a one byte wide 100K endless loop memory. The block diagram of Figure 4 presents this system design. Table IV lists its characteristics. This design incorporated several requirements in its design philosophy. The goal was to create a unit which was as simple as possible, consumed very little power and which could be constructed for a mimimum of cost. As a consequence a minimum of control functions were incorporated in the design and no housekeeping or addressing registers were used. Instead, an interface which correlates directly to the bubble memory functional elements is presented to the interface to be driven by the controlling device. The four basic commands delivered to the interface correlate exactly with the basic memory function. These are: (1) the run/stop command; (2) the erase command; (3) the write command; and (4) the read command. These four functions, controlled in combination by the parent equipment are sufficient to exploit the full capabilities of the memory module. To obtain minimum power, each control line power strobes the section of circuitry needed for that function, otherwise each section is kept off. When awaiting a request for a read or write the recorder power is limited to 30 mw which is consumed in the oscillator section. In one relatively easy implementation, the 8-10^5 bit chips are treated exactly as an endless tape loop. The data is divided into a series of records as shown in Figure 5a. Each record block length is assigned as necessary, and header information at the beginning of each block sufficient for identification is written. With this organization, the information contained in the memory module is suffi-

TABLE II—Access Times

TTL RAM	60 ns
MOS RAM	300 ns
CORE	500 ns
BUBBLES (FAST AUXILIARY CONFIG.)	0.5-1 ms
HPT DISK/DRUM	8 ms
MOVING HEAD DISK	50 ms
FLOPPY DISK	100 ms
BUBBLES (ENDLESS LOOP CONFIG.)	1 sec
CASSETTES	10 sec
TAPES	10 sec

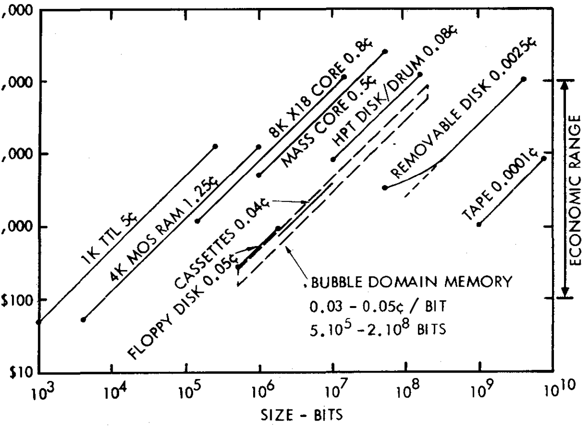


Figure 1—Cost-size comparisons

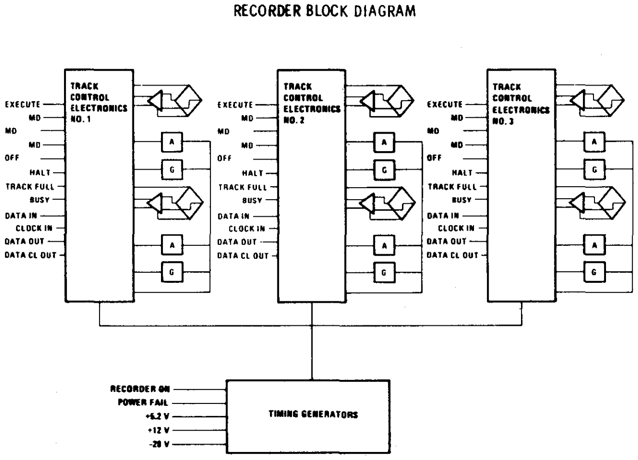


Figure 2—Feasibility model block diagram

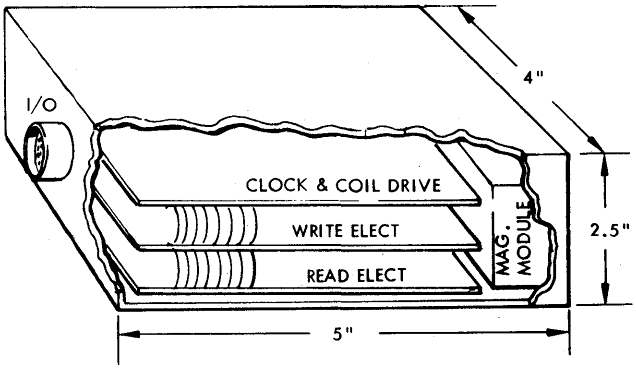


Figure 3—Endless loop recorder

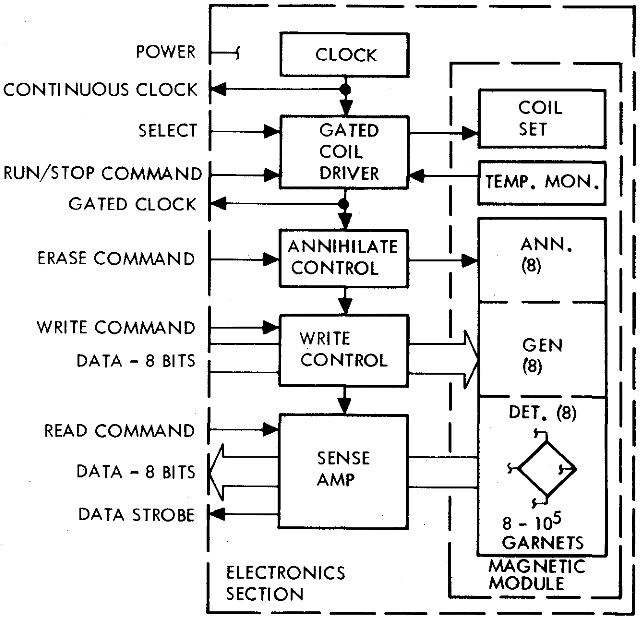


Figure 4—Endless loop recorder block diagram

TABLE III—Feasibility Model Characteristics

SIZE:	3 Tracks of 20,000 Bits (6×10^4 Bits)
DATA RATE:	DC-150K Bits/Sec—Each Track Independent
AVERAGE SEEK TIME:	Not applicable
ADDRESSING:	None—Total Track Read

TABLE IV—Endless Loop Recorder Characteristics

SIZE	100,000—8 Bit Words (8×10^5 Bits)
DATA RATE:	50K Words/Second (4×10^5 B/Sec.)
AVERAGE SEARCH:	1 Second
ADDRESSING:	Header Data Comparison
BLOCK LENGTH:	As Desired

cient for a record search and read function. Figure 6 illustrates a representative controller which can couple the endless loop recorder to a computer bus structure. In this configuration, the controller provides the necessary mode control to search the bubble memory for the proper header, read or write data as required, and provide the interface with the computer and its main memory. Use of this interface is envisioned mostly as a means by which data could be dumped for processing. The original recording could take many other forms. For example, the memory could be coupled to a keyboard for manual data entry or the input could be composed of the simple recording of data from a variety of sensors. This design is intended to be especially useful in interactive modes where access time is not important, as with a human

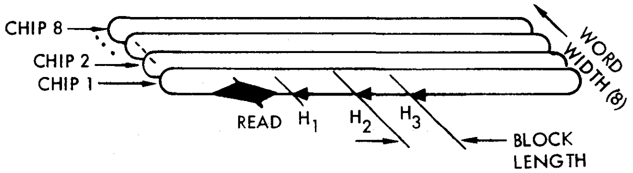


Figure 5a—Endless loop data organization

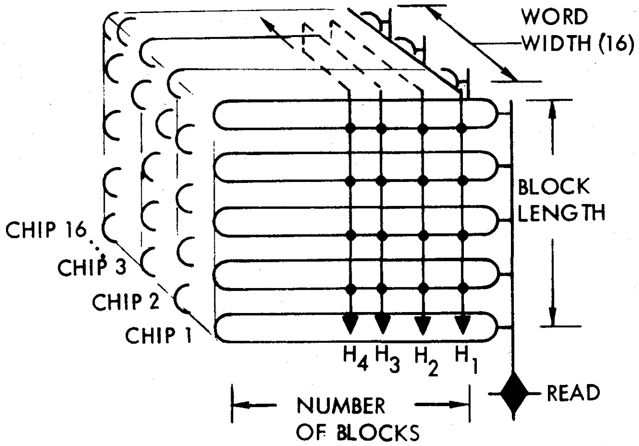


Figure 5b—Fast access memory data organization

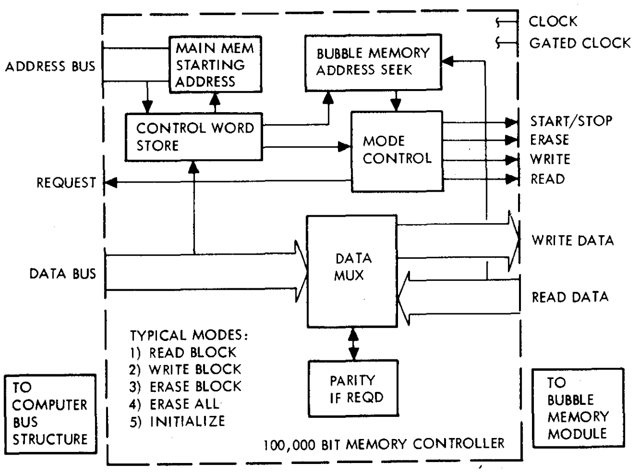


Figure 6—Representative controller

operator or some other device like microprocessors which require non-volatility and a low data rate.

FAST AUXILIARY MEMORY

In direct contrast to the simple endless loop memory, the Fast Auxiliary Memory is a relatively large rack mount device designed for high speed continuous operation. This unit is intended to serve as a non-mechanical equivalent to a head-per-track disc/drum while exhibiting an access time an order of magnitude faster. Table V lists its characteristics. The memory is modular in design with a single control module capable of addressing as many as sixteen memory modules on a bus system. This is analogous to the board expandable core memory systems where a customer is allowed to choose an initial memory size and increase capacity later as desired. Figure 7 illustrates the block diagram of the fast auxiliary bubble memory system. This memory is somewhat equivalent to a multiple spindle disc memory except that each spindle is momentarily stopped and awaiting a command to jump forward to the next piece of required data. A significant difference between the endless loop recorder and the fast access memory system is the reorganization of the bubble memory chip to achieve the access time required. Figure 8 diagrams the organization of the chip. A series of small loops is used to store the data rather than a single large loop. Access is achieved by rotating data in the minor loop until that record which is desired is opposite the replicator. At that time, the record is imaged to the gathering path and delivered to the bubble memory detector. Fast access is achieved due to the small size of the minor loop and the consequently small number of field rotations required to bring any desired block up for reading. The use of many chips as illustrated in Figure 6b is the application actually used in the FAM module. Sixteen chips running simultaneously make the block of data a series sixteen bit words.

TABLE V—Fast Access Memory Characteristics

SIZE:	65K—16 Bit Words (1.05 × 10^6 Bits) to 1048K—16 Bit Words (1.67 × 10^7 Bits)
DATA RATE:	150K Words/Sec. (2.4 × 10^6 B/S)
AVERAGE LATENCY:	.986 ms
QUEUED LATENCY:	.133 ms
BLOCK LENGTH:	128 Words to 16K Words in 128 Word Increments

The addressing technique used in the Fast Auxiliary Module varies significantly from that used in the endless loop recorder. In the recorder, the seeking of a record data requires that all records between the starting position and the desired record be transversed in finding the required data. As a consequence, the reading of the header tags during this transversal can be used as the search method for finding any particular record. No access time penalty is paid for looking as-you-go. In the fast auxiliary module however the rotation of the minor loops until the desired

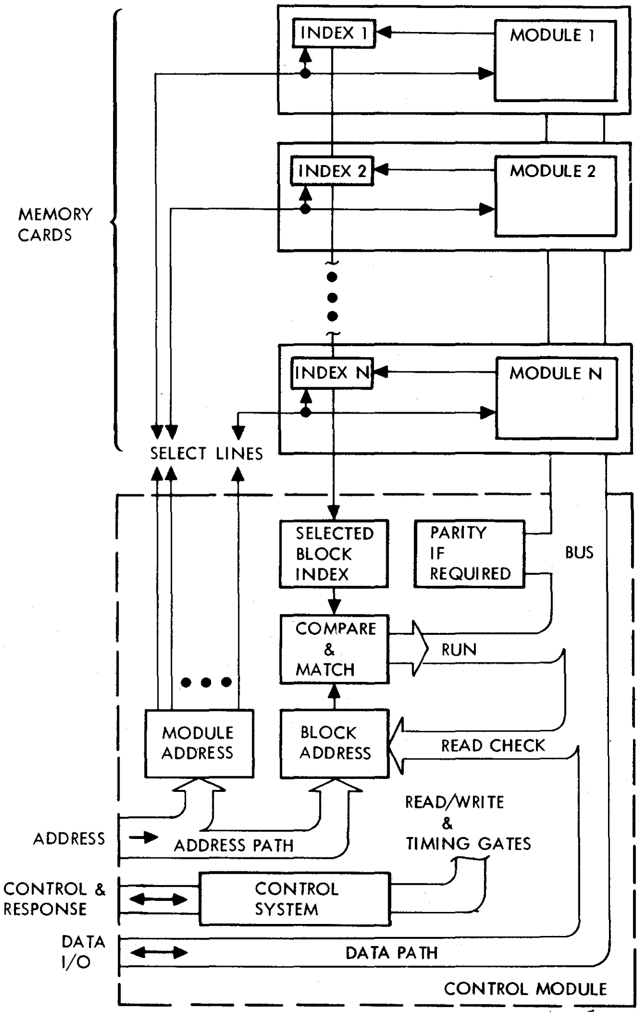


Figure 7—Fast access memory system diagram

record is brought up implies that none will be brought out prior to that which was addressed. This direct indexing to the record sought without reading intermediate headers requires that address information external to the memory media must exist in the system. In the block diagram an address register is shown in each module corresponding to the position at which each memory is resting. From this address and the request address, the number of field rotations required to bring up the desired data block can be derived. This auxiliary address store is envisioned as volatile. There is no need for it to be non-volatile as a single interrogation of a memory board for an initialization mode can recover the address information from the head of the block nearest the replicator switch.

Another difference between the fast access module and the endless loop module is the apparent dependence of the block size upon the number of minor loops incorporated in the chip design. In an endless loop module, any number of steps along the path can be assigned to a record. Consequently no implicit block size exists. In the fast access module, the block size appears to be tied directly to the number of minor loops. This is not necessarily true. As it turns out, the delivery of a single block of data from the minor loops can be immediately succeeded by successive blocks. The delivery of successive blocks with no gap in between allows the fast access module to deliver combined blocks of any desired length. By designing a minor loop chip so that the number of bits transferred into the main track is prime with respect to the number of records positioned in the minor loop a situation can be established where the entire content of the chips can be read in rapid succession with one block immediately following the preceding until every block has been read. After the last block, the first immediately reappears. Thus with this organization of the minor loops, the chip can be made to emulate a single continuous loop. Rapid access to any portion of the loop can be made by merely bringing the proper section of data to the replicate switches before the dump sequence begins. The "mutually prime" minor loop design allows block sizes which are multiples of the minimum block size.

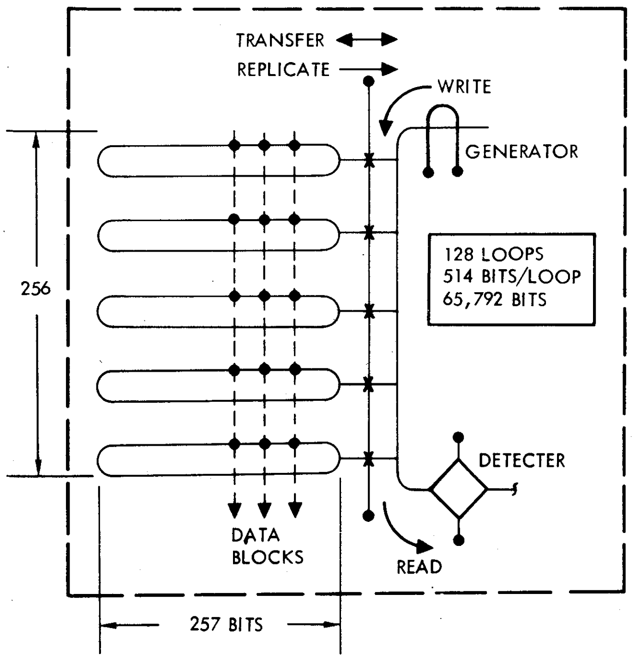


Figure 8—Minor loop chip with replicate read

Queuing of records for absolute minimum access time can be achieved by bringing the memory to the desired address and stopping at that point prior to the actual request for data. Often when data is stored in successive records this look-ahead will naturally occur due to the start/stop nature of the bubble memory. When the reading of a given record is complete, the memory stops awaiting the next command. When the sequence requires the reading of successive records each is immediately available.

CONCLUSION

The work for the last two years which has culminated in the building of prototype bubble domain memory systems has established the technology. Credible systems which do indeed exhibit the predicted bubble domain features have been built. The use of bubbles in the 4 and 6 micron diameter range for useful systems is established. These bench marks established, the drive is now to produce systems tailored to the user needs. The two examples given in this paper address those needs. Initial production of units at this point is expected. Additional fundamental developments of garnet material in a 1 and 2 micron bubble size will expand the basic capabilities available for bubble memory systems. This expansion is useful and insures a future growth of bubble memory. However, the 4 and 6 micron bubble sizes are suitable for commercial exploitation.

ACKNOWLEDGMENTS

The author wishes to acknowledge several of his associates at Rockwell International who have participated in the work which has brought these memory systems to fruition. First is John Archer who has headed up the entire program. Oliver D. Bohning and T. T. Chen were the principal developers of the NASA feasibility recorder. Thomas M. Steury was responsible for many of the system concepts incorporated in the fast access memory design. Finally, the author also wishes to thank William C. Mavity for his many suggestions and kind assistance in the preparation of this paper.

BIBLIOGRAPHY

1. Ypma, J. E., "Magnetic Bubble Memories," National Aerospace Electronics Conference, Dayton, Ohio, May 1974, pp. 50-54.
2. Chen, T. T., O. D. Bohning, L. R. Tocci, J. L. Archer, and R. L. Stermer, "A Magnetic Bubble Domain Flight Recorder," IEEE Trans Intermag 1974, Toronto May 1974, MAG 10, No. 3, pp. 739.

3. Bailey, R. F., and J. P. Reekstin, "Yield Analysis of Large Capacity Magnetic Bubble Circuits with Redundancy Design," *IEEE Trans Intermag 1974,* Toronto May 1974, MAG 10, No. 3, pp. 856.
4. Archer, J. L., L. R. Tocci, O. D. Bohning, D. H. Baird, C. F. Buhrer and J. J. Vytal, "Reliability of Magnetic Domain Memories," 19th Conf. on Magnetism and Magnetic Material, Boston, Nov. 1973, paper 3A-2.
5. Chen, T. T., J. L. Archer, R. A. Williams and R. D. Henry, "Radiation Effects on Magnetic Bubble Domain Devices," *IEEE Trans. MAG 9,* No. 3, pp. 385-9 (1973).
6. Bobeck, A. H., "Magnetic Bubble Domain Devices," *Intermag Conf.,* Denver, Colo. 1971.
7. Rifkin, A. A., "A Practical Approach to Packaging Magnetic Bubble Devices," *IEEE Trans. Magnetic MAG 9,* 429, 1973.
8. *IEEE Transactions on Magnetics,* Sept 1973, Vol MAG 9, No. 3 Intermag Papers—Sessions 4, 13, 17, 21, 26 & 29.
9. Feth, G. C., "Memories are Bigger, Faster—and Cheaper," *IEEE Spectrum,* November 1973, pp. 28-35.
10. Brooks, F. P., Jr., "Mass Memory in Computer Systems," *IEEE Transactions on Magnetics,* Sept. 1969, Vol. MAG 5, No. 3, pp. 635-639.
11. Matick, R. E., "Review of Current Proposed Technologies for Mass Storage Systems," *Proceedings of the IEEE,* Vol 60, No. 3, March 1972, pp. 266-289.
12. *Report of Tape Recorder Action Plan Committee,* March 21, 1972, NASA (National Tech Inf. Service, Springfield, VA).
13. Mavity, W. C., and J. P. Davis, "Applications of Magnetic Bubbles," *1972 WESCON Technical Papers*—Session 8.
14. Mavity, W. C., "Bubble Memory Status and Trends," *1972 Government Microcircuit Applications Conference,* (GOMAC) pp. 420-426.
15. *Electronic Design 22,* October 25, 1974, "The Great Memory Battle Goes on, but Semiconductors Appear the Ultimate Victors," pp. 40-48.
16. Chang, Hsu, "Capabilities of the Bubble Technology." *National Computer Conference,* 1974, pp. 847-855.
17. Carson, R. W., "Minicomputer Removable Storage," *Modern Data,* March 1973, pp. 46-50.

Superconducting memories employing Josephson devices

by W. ANACKER
IBM Corporation
Yorktown Heights, New York

INTRODUCTION

Experimental superconducting Josephson devices being investigated for use in digital logic and memory circuits have been demonstrated to switch in the 10 to 100 picosecond (1 picosecond = 10^{-12} sec) range. Projections based on the operation of individual logic circuits indicate that they may surpass semiconductor circuits in very high performance CPU's. This potential is based on the fact that these circuits dissipate extremely little energy, on the order of 10 to 1000 attojoules (attojoule = 10^{-18} joule), while operating with subnanosecond delays. It should, therefore, be possible to package Josephson devices very densely and interconnect them by properly terminated superconducting transmission lines so that the fast switching speed of individual circuits is retained throughout large logic networks.

Ultrafast CPU's need, of course, suitable memories, which are fast enough to provide requested data quickly and which are large enough to ensure that high throughput is sustained. It is likely that memory hierarchies will be needed to satisfy these requirements. Improved performance of the individual memories forming these hierarchies will, however, be required also, to avoid excessive "tuning" of such hierarchies, i.e., good performance for only narrow classes of computational tasks. The potential of Josephson devices to provide such memories and measures to "detune" such hierarchies will be discussed in this paper.

JOSEPHSON DEVICES AND CIRCUITS

Devices

Based on superconductivity and electron tunneling effects discovered in the early 1960's by Josephson[1] and Giaever,[2] one may construct a thin film switching device as shown in Figure 1a. Two thin film strips of superconducting material, e.g., Pb, are deposited in a partially overlapping arrangement but separated from each other by an extremely thin "native" oxide tunnel barrier of about 30 Å (1 Å = 10^{-8} cm) thickness. A small voltage of a few millivolts will cause a tunneling current of single electrons through the insulating oxide barrier by virtue of the quantum mechanical tunnel effect. The current-voltage relation is linear, as shown in the current-voltage plot of Figure 1b, as long as the temperature T of the structure is above the critical temperature T_c of the film strips (in case of Pb films $T_c = 7.2°\text{K}$). One may represent this behavior by a voltage independent so-called normal tunnel conductance.

If the structure of Figure 1a is immersed in liquid helium, which maintains a temperature of 4.2°K under atmospheric pressure, both film strips are superconducting and the current-voltage relation undergoes two characteristic changes, as shown in Figure 1c. First, a supercurrent,[3] made up of paired electrons (Cooper pairs) which are formed below T_c and are responsible for all superconducting effects, can flow through the oxide barrier up to a well defined dc-Josephson current threshold I_m without causing any voltage drop ($V=0$) across the oxide barrier. Second, the current-voltage relation becomes non-linear, i.e., the tunnel conductance becomes voltage dependent. In particular, the tunnel current is strongly suppressed at voltages $V<V_g$, it surges up at $V=V_g$ and it approaches the "normal" tunnel current at $V>V_g$. Here, $V_g = 2\Delta/e$ denotes the gap voltage ($V_g =$ 2.5 mV for Pb), 2Δ the superconducting energy gap of the film strips and e the electronic charge.

Apparently, the thin film structure of Figure 1a, when immersed in liquid helium, behaves either as a superconductor ($V=0$) or as a tunnel conductance ($V\neq 0$) over a substantial range of current. Figure 1d denotes a further property of this structure, namely, a dependence of the dc-Josephson current threshold I_m on magnetic flux Φ enclosed in the junction. This dependence is more clearly shown in the I_m vs Φ plot of Figure 1e. One may exploit this property by providing a third thin film strip on top of the junction and insulated from both junction electrodes. A "control" current I_c flowing through this third line generates a magnetic flux penetrating the junction and thus modulates the threshold I_m. Accordingly, one may relabel the horizontal axis of Figure 1e by I_c and interpret the plot as follows: if the vector sum of junction and control currents falls inside of the area under the curve, the voltage across the junction is zero; if the vector falls outside of the area, the voltage is finite. The functional dependence of $I_m(I_c)$ can be varied widely and predictably by engineering design.

The transition time from $V=0$ to $V=V_g$ is predominantly governed by the junction capacitance C and the charging current I_g through the junction and is given approximately by

$$t_R = C(V_g/I_g) \qquad (1)$$

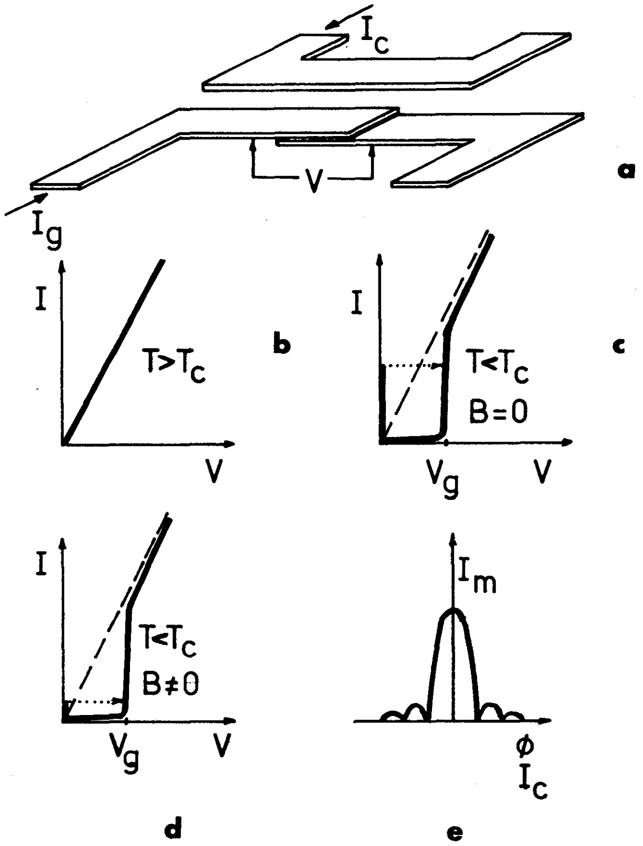


Figure 1—Josephson tunneling device (a) Structure (b, c, d) Current-voltage plots (e) Threshold current-flux (and control current) plot

Since, for miniaturized junctions, C becomes quite small and V_g is small anyway, transition times of 10 to 50 psec are readily obtainable.[4,5] It should be noted that the transition back to $V=0$ occurs usually at a current much less than I_m, i.e., the device exhibits hysteresis,[6] and the transition proceeds somewhat more slowly. Extensive reviews of Josephson devices can be found in References 7 and 8.

Circuits

Most circuits with Josephson devices which have been investigated so far are of the basic configurations shown in Figures 2a and 2b. Either Josephson devices are incorporated in both branches[9] of a superconducting loop (Figure 2a) or one Josephson device is shunted via superconducting striplines with a resistor[10,11] (Figure 2b). External current supplied to the loop of Figure 2a can be routed through either branch by forcing the Josephson device of the alternate branch into the $V\neq 0$ state. The device will automatically revert to the $V=0$ state when all (or almost all) current has been rerouted. Except during actual rerouting of current, no power will be dissipated in this circuit. It is noteworthy that, once all incoming current is routed into one branch, the device in the other branch has reverted to $V=0$, and when the external current is then switched off, a circulating current is established in the loop to maintain the magnetic flux linked with the totally superconducting loop. The circulating current is persistent, does not dissipate power and can be used to store binary information. The time required to reroute current is governed by the inductance L of the loop, the driving voltage V_g and the amount of current I to be rerouted. It is approximately given by:

$$\Delta t = L\left(\frac{I}{V_g}\right). \qquad (2)$$

The incoming current in the configuration of Figure 2b will pass totally through the Josephson device as long as it is in the superconducting state. When it is switched to $V\neq 0$, part of the incoming current (i.e., V/R) will be diverted to the resistor path. The resistor value may be chosen to intersect the current-voltage characteristic of the device at $V=V_g$ or at $V<V_g$ and, in particular, such that the circuit is "latching"[10,11,12] i.e., needs resetting to $V=0$, or "non-latching," i.e., follows the input control signals as, for example, described in Reference 13. As indicated in Figure 2b, more than one control line may be placed on top of the junction and be used by proper design to perform AND and OR logic functions.[12] Clearly, networks can be assembled with circuits as shown in 2a and 2b by using the output striplines of devices to control other devices.

The characteristic impedance of the striplines connecting Josephson device and resistor can be chosen such that the resistor represents a matched termination, thus providing a zero reflection factor. In this case, the time needed to establish the output current at the location of the resistor is governed by the voltage risetime and the propagation delay of the striplines.

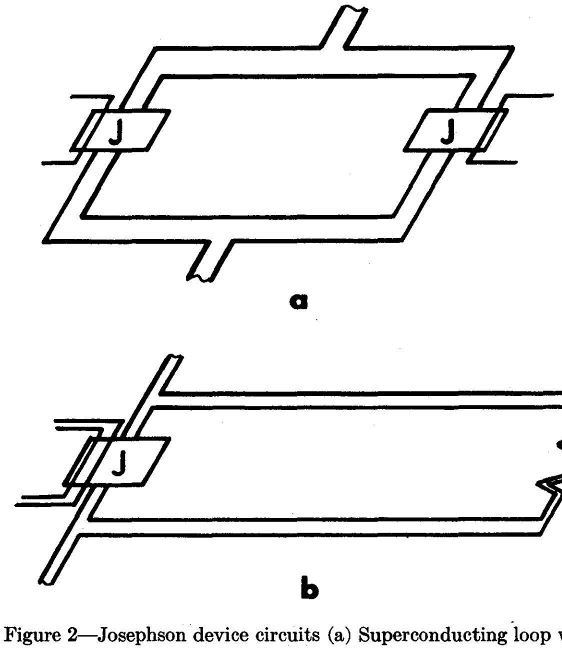


Figure 2—Josephson device circuits (a) Superconducting loop with two devices (b) Resistive loop with one device

Hence

$$\Delta t = t_R + t_d \tag{3}$$

where t_R denotes the risetime at the device and t_d the delay in the lines. An average logic delay of 200 psec per circuit has been demonstrated in a 1 bit full adder[14] controlled with Josephson devices.

MEMORY COMPONENTS

NDRO Loop Cells

A practical NDRO memory cell[15,16] for bit organized random access operation is shown in Figure 3. It comprises a superconducting loop with Josephson devices A and B in each branch, both controlled by "control" bit lines for writing; one branch of the loop acts as control for a third Josephson device S for non-destructive read out. Persistent circulating currents ($I_w/2$) in clockwise or counterclockwise directions represent stored binary ones and zeros, respectively. The conservation of magnetic flux, for as long as the loop remains totally superconducting, dictates that when an external current I_w is applied, it must split equally into both branches and superimpose on the circulating current such that I_w flows through one branch and zero current through the other; it also dictates that the original circulating current is restored when the external current is switched off. Thus, the direction of circulating current can be detected without disturbing the information by applying a word current I_w which causes either I_w or zero current to flow in the branch controlling device S and a sense current I_s which causes the device S to switch only when its control current exceeds $I_w/2$. Data is written into a cell by coincident word current I_w and bit current I_B which can switch one device in the selected cell and reroute the word current into the alternate branch.

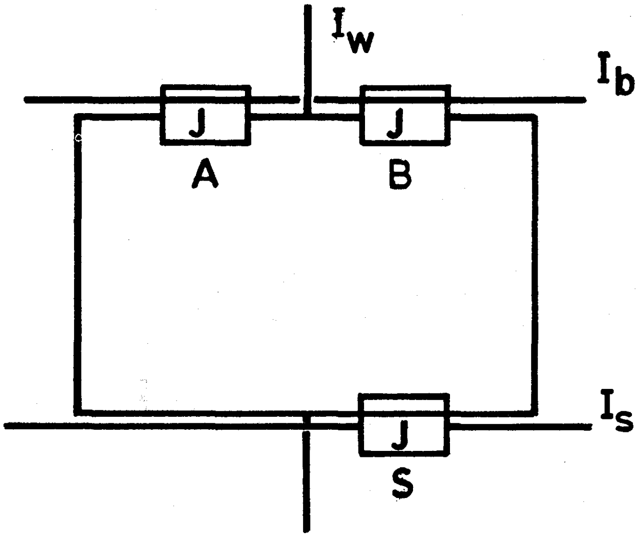


Figure 3—NDRO random access memory cell

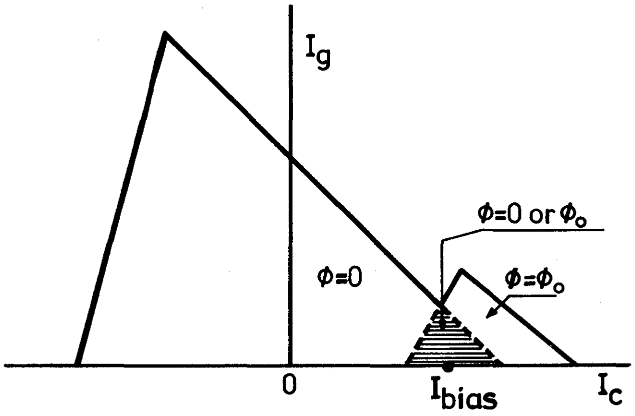


Figure 4—Gate current-control current plot of single device memory cell

Experiments have demonstrated that data can be written in rather large memory cells[16] with 2 mil minimum line width and ~250 mil² area in about 600 psec and in miniaturized cells[17] with 2μ minimum line width and ~1.4 mil² area in less than 100 psec. It was also demonstrated that more than $5 \cdot 10^8$ NDRO operations left the stored information undisturbed.[16]

DRO Single Device Cells

Single specially shaped or elongated Josephson devices[18,19] can be used for data storage in a DRO bit organized random access mode as well. These devices admit magnetic flux in discrete quanta of magnitude $\Phi_o = h/2e \sim 2 \cdot 10^{-15}$ V-sec, where h denotes Plank's constant and e the electronic charge, when their control currents are steadily increased; they possess an operating region in the $I_g - I_c$ plane as shown shaded in Figure 4 in which either 0 or 1, flux quanta can be maintained without standby power and stably by a suitable bias control current. The number of flux quanta actually stored in such devices can be increased (decreased), from 0 to 1 (1 to 0) by a temporary increase (decrease) of the bias control current. The presence of a flux quantum can be detected either by the occurrence of an energy spike[18] upon reduction of the bias control current if a flux quantum was stored or by the modification[19] which a stored flux quantum exerts on the dc-Josephson threshold current I_m. In both cases, the stored information is destroyed and must be rewritten. The energy associated with a flux quantum which is maintained in a device, for example, by a circulating current of 1 to 10 mA amounts to about 2 to 20 attojoules. Although this energy is extremely small, even fractions of it have been detected experimentally by Josephson device detectors.[18] This is in part due to the fact that the release of a flux quantum as found by simulation leads to an energy spike of only a few picoseconds duration, but a peak of a few hundred microvolts and microamperes. The expected advantage of single device cells is their potential for large memories with high bit densities.

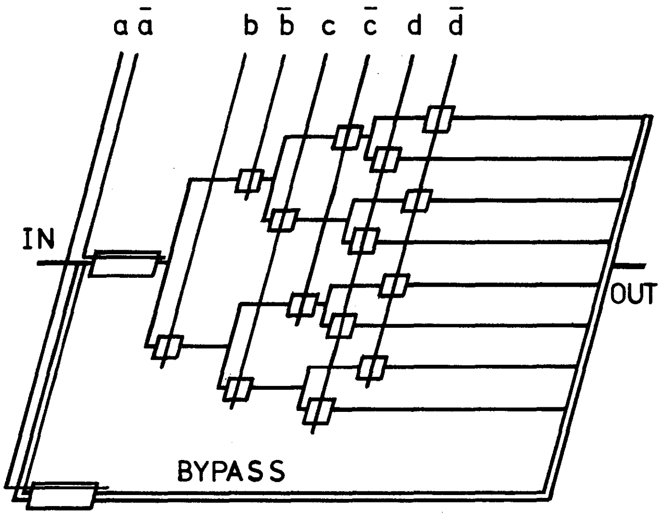


Figure 5—Tree decoder

Decoders and Drivers

Addresses can be decoded with Josephson devices by tree[15] decoders. Tree decoders consist of branching networks with one or more Josephson devices in each branch, as shown in Figure 5. The branches of the last stage can be connected either directly or via drivers to the array lines. The Josephson devices in each branch are controlled by the "true" and "complement" outputs of address registers. For each address, all but one branch through the decoder tree contain Josephson devices in the $V\neq0$ state, while all Josephson devices along the selected branch remain in the $V=0$ state. It is convenient to provide a bypass line to carry the current around the array when desired. The decoder and bypass configuration can be supplied with dc current. Apparently, any selected branch and array line form a "giant" superconducting loop with the bypass line, where current rerouting is driven by the one or more Josephson devices in the bypass line. The time needed to route current from the bypass into a selected array line is then given by Equation 2. The operational speed can be increased and residual disturb currents in nonselected array lines eliminated by using line drivers comprising Josephson devices which are controlled by output branches of a tree decoder; their output lines are connected to array lines. Thus, the tree decoder loops are minimized and speed is gained when the output (array) lines of the drivers are terminated in their characteristic impedance. The total delay time is then the sum of (reduced) current routing time through the tree decoder according to Equation 2 and the signal propagation time through the array line according to Equation 3.

Sense Signal Detection

Sense gates of NDRO loop cells (see Figure 3) can be interconnected into rows of sense lines[15] which are provided at one edge of the array with superconducting bypass lines, as shown in Figure 6. These bypass lines can control, in turn, a column of Josephson devices outside of the array which, via another bypass line, can finally control a single read out Josephson device for the whole array. In this arrangement, the read out time, beginning from the time when a sense gate in the array has been switched, is given by the sum of two current routing times according to Equation 2.

Of course, it is possible, in principle, to terminate sense lines and the "column" line in their own characteristic impedances also, in which case the read out time would be the sum of signal propagation times according to Equation 3 for one row and the "column" line. The practicality of this approach depends on tolerance and margin considerations and can be assessed only in a realistic and detailed design.

DRO single device cells generate their read out signals on that array line on which they are serially strung. In coincidence operation, this line will carry a current pulse for cell selection. The read signal must, therefore, be separated from the drive pulse during read out. Special strobed read detection circuits are required if the energy spike upon release of a flux quantum is to be detected. That this can be done in principle has been experimentally demonstrated.[18] If, on the other hand, the modification of the threshold current I_m by a stored flux quantum is used for read out,[17] either one of the schemes described in conjunction with the NDRO loop cells can be employed for read out.

LSI FABRICATION

Since Josephson devices can perform memory, drive, detection and decode functions, integration of arrays and peripheral circuits on a common substrate and—drawing on experience from semiconductor technology—adoption of a wafer and chip fabrication concept[20] with photolithographic techniques for pattern definition and evaporation and sputtering methods for building a multilayered structure are quite natural. Once large scale integration has been adopted, the fabrication process must allow for material and process compatibility of all required circuit components, i.e., Josephson devices, resistors, superconducting interlayer contacts,

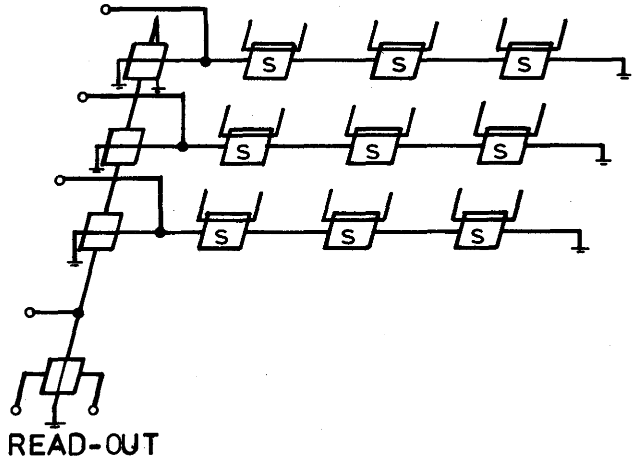


Figure 6—Sense detection scheme

insulation and striplines. A major challenge in this respect, of course, is the reproducible preparation of extremely thin and pinhole-free native oxide tunnel barriers on the surface of thin metal films which have been subjected to photolithographic chemistry and ambient atmosphere with contamination being unavoidable. An RF sputter technique[21] in which growth rates of oxidation and removal rates of sputter etching are balanced has been developed and has so far provided encouraging results as to the thickness reproducibility of thin oxide tunnel barriers.

PERFORMANCE PROJECTION

To date, memories of practical size have not been built or designed. Performance must, therefore, be projected by estimating cell and array sizes and cycle and access times on the basis of the experimental circuits and the delay equations mentioned above. Since LSI has been adopted, it is all but impossible to predict ultimate bit densities and access times because they depend strongly on line width resolution which, in turn, is likely to improve as a result of present exploration of uv, *e*-beam and x-ray exposure techniques. Just how much improvement is possible is not clear yet, however. Potential performance of Josephson device memories with present state-of-the-art resolution of, say 0.2 mil minimum line width—i.e., what could be expected if Josephson device technology were developed and ready for manufacturing—is estimated instead.

On the basis of the previously mentioned 1.4 mil^2 loop cell with 2μm wide lines and an upper bound of switching time of 100 psec, an NDRO loop cell with 0.2 mil wide striplines may be estimated to occupy an area of about 2.5×2.5 mil and to switch in about 100 psec. Then, an array of 64×64=4096 cells of NDRO loop cells should fit into an area of about 160×160 mil. With peripheral circuits being located on the same chip, one may make the following assumptions for such an array: word, bit and sense lines, tree decoder branches, matrix decoder control lines and a sense column line are all about 160 mil long and 0.2 mil wide. The loop inductance for those striplines (with 2000 Å thick insulation) would amount to $L \sim 350$ pH, the propagation delay to $t_d \sim 100$ psec and the risetime of the output of a Josephson device driver to $t_R \sim 50$ psec. It is also assumed that drive currents of about 10 mA and sense currents of about 2.5 mA are used and that four serially connected Josephson devices per decoder branch and bypass line are provided.

Taking note finally of the pulse sequences required for bit organized NDRO random access and allowing for sufficient pulse overlap, access and cycle times on the order of 2 to 2.5 nsec are derived for a low power, 4K bit random access NDRO array with a bit density of about $1.6 \cdot 10^5$ bits/inch^2.

DRO single device cells are likely to occupy less area than NDRO loop cells since only one Josephson device and no loop is required. Four times as many cells could possibly be fitted in the same array area. Total access and cycle times would likely be longer due to the need for extra pulses to perform rewriting and the possible need for sense signal amplification. However, even if access and cycle times would turn out to be an order of magnitude longer than those derived above, the potentially high bit density of about $6.4 \cdot 10^5$ bit/inch^2 combined with low power and 20 to 30 nsec access and cycle time would appear quite attractive for use as main or bulk memory in hierarchies to feed ultrafast CPU's.

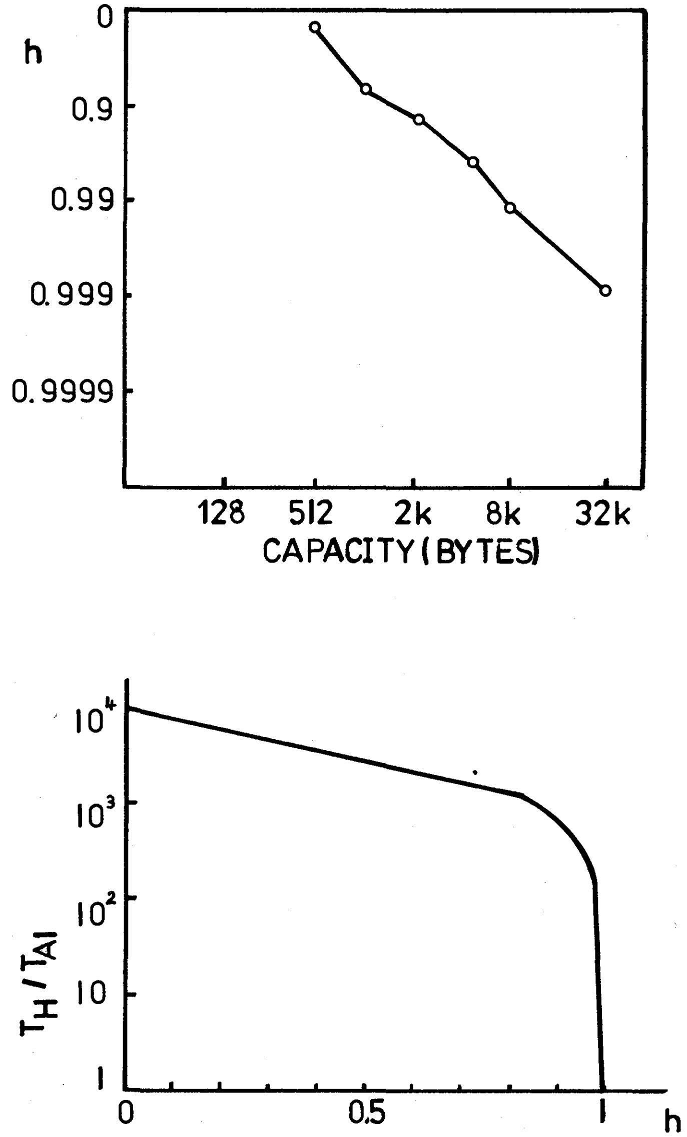


Figure 7—(a) Hit ratio h versus capacity C_1 plot (b) Hierarchy access time ratio T_H/T_{A1} versus hit ratio h plot

A NOTE ON MEMORY HIERARCHIES

Although the memory hierarchy concept has been proven to approximate the ideal of a large *and* fast memory quite well, it is noteworthy that certain drawbacks are associated with the concept as well. Memory hierarchies tend to be "tuned" to specific classes of computational tasks and moreover, the sharpness of the "tuning" curve depends on the access time ratio of the hierarchy's memories and their capacities. To elucidate this point, consider first the effective hierarchy access time T_H of a two level hierarchy with memories M_1 and M_2.[22] T_H is a function of the so-called hit ratio h and the access time ratio of T_{A2}/T_{A1} where T_{A1} and T_{A2} denote the access times of memory M_1 and M_2 respectively. The hit ratio h is an empirical measure defined by the ratio of requested data found in M_1 over the total number of data requests issued by the CPU. It is clearly a measure of dynamic clustering of data addresses in address space. It depends, however, also on the capacity of the faster and smaller memory (here M_1) of the hierarchy, as indicated in the plot of Figure 7a,[22] which shows that h increases for increasing C_1. The hierarchy access time T_H as a function of h for a rather large ratio of $T_{A2}/T_{A1}=10^4$ is plotted in Figure 7b. This plot signifies that T_H drops significantly only when the bit ratio h exceeds 0.95 to 0.99. Clearly, this hierarchy would perform well only for a set of tasks with large hit ratios h. One can "detune" this hierarchy by (a) providing memories M_1 and M_2 with a more favorable, i.e., smaller, T_{A2}/T_{A1} ratio and (b) providing larger memory capacity C_1.

In the case of Josephson device memory hierarchies for ultrafast CPU's, the following strategy might be suggested: since M_1 with access time T_{A1} on the order of 3 nsec is so fast that considerations of signal delays through the memory package will likely be limiting its capacity C_1, it is advisable to reduce the access time T_{A2} of M as much as possible to obtain a favorable ratio T_{A2}/T_{A1} of, say, 10 to 20. The memory M_2 will likely be backed up by conventional memory or storage device levels M_3 with a rather large access time ratio T_{A3}/T_{A2} being unavoidable. Therefore, the capacity C_2 of M_2 should be made as large as feasible to increase h. In consequence, one should focus on making M_2 as large and as fast as possible.

SUMMARY

Josephson devices, circuits and memory components have been reviewed. An estimate of potential memory performance with state-of-the-art photolithographic resolution has been made, and it is found that rather high performance buffer and main memories for ultrafast CPU's can be envisioned. It is indicated that considerations of memory hierarchy performance favor the provision of a main memory with large capacity and access times on the order of 20 to 100 nsec. It is believed that the performance potential of Josephson device memories and CPU's warrants further investigation of the feasibility of this new technology.

REFERENCES

1. Josephson, B. D., *Phys. Letts.* 1, p. 251, 1962.
2. Giaever, I., *Phys. Rev. Letts.* 5, p. 147, 1960.
3. Bardeen, J., L. N. Cooper and J. R. Schrieffer, *Phys. Rev.* 108, p. 1175, 1957.
4. Zappe, H. H., and K. R. Grebe, *J. Appl. Phys.* 44, p. 865, 1973.
5. Jutzi, W., Th. O. Mohr, M. Gasser and M. P. Gschwind, *El. Letters* 8, November 30, 1972.
6. McCumber, D. E., *J. Appl. Phys.* 39, p. 3113, 1968.
7. Matisoo, J., *IEEE Trans. MAG-5*, No. 4, p. 848, 1969.
8. Solymar, L., *Superconductive Tunneling and Applications*, Wiley-Interscience, New York, 1972.
9. Matisoo, J., *IEEE Trans. MAG-5*, No. 4, p. 848, 1969.
10. Anacker, W., *Proc. of FJCC*, p. 1269, 1972.
11. Henkels, W. H., *Trans. IEEE MAG-10*, p. 860, 1974.
12. Herrell, D. J., *IEEE Jour. of Solid State Ckts*, SC-9, p. 277, 1974.
13. Baechtold, W., T. Forster, W. Heuberger, and Th. O. Mohr, *IBM* RZ657, 1974.
14. Herrell, D. J., *Trans. IEEE MAG-10*, p. 862, 1974.
15. Anacker, W., *Trans. IEEE MAG-5*, p. 968, 1969.
16. Zappe, H. H., to be published, *IEEE Jour. of Solid State Ckts.*, Feb. 1975.
17. Broom, R. F., W. Jutzi, and Th. O. Mohr, *Applied Superconductivity Conference*, 1974.
18. Gueret, P., *Appl. Phys. Letts.* 25, p. 426, 1974, and paper M-2 *Applied Superconductivity Conference*, 1974, to be published in *IEEE Trans. MAG*, March 1975.
19. Zappe, H. H., *Appl. Phys. Letts.* 25, p. 424, 1974.
20. Greiner, J. H., S. Basavaiah, I. Ames, *J. Vac. Sci. and Tech*, 11, p. 81, 1974.
21. Greiner, J. H., *J. Appl. Phys.* 42, p. 5151, 1971.
22. Anacker, W., *IEEE Trans. MAG-7*, No. 3, pp. 410-415, 1971.

Holographic memories—Fantasy or reality?

by A. K. GILLIS, G. E. HOFFMANN and R. H. NELSON
Harris Corporation
Melbourne, Florida

INTRODUCTION

Twelve years have passed since Leith's historic paper[1] which opened the era of modern holography. Although initially investigated for its unusual imaging properties, we have seen about five years of intensive research on the application of holography to digital data storage. Because of its unique properties, it is not surprising that attempts have been made to apply holography to such a broad range of memory and storage hierarchy. Activity has ranged from developing small-capacity, high-speed memories to large-capacity, read-only storage in the multiterabit range. Additionally, significant activity has been directed toward solving the highly specialized problems associated with ultra-high data rate recorders and reproducers. Memory systems are now, and will continue to be, the highest single cost item in the computer hardware structure. This, at least in part, accounts for the intensive research activity in optical alternatives to computer memory and storage.

What is the current status of our research and in what segments of the memory hierarchy are holographic techniques likely to play a significant role? Although research continues across the broad spectrum of memory hierarchy, some strong indicators point to very specific areas where the technology has a reasonable chance of success.

Before we consider the specific nature of holographic memories, we review the current state-of-memory technology and identify the targets at which holographic memories have been aimed. Perhaps the two most widely used performance measures for memories are capacity and access time. Clearly there are many other factors such as transfer rate, size, power consumption, interface ease, reliability and reproducibility which may play equally important roles in characterizing memory performance. Similarly, memory cost or more commonly, cost per bit, forms one of the important criteria for memory selection. For purposes of this discussion, capacity and access time will be sufficient factors if we remember that even the ultimate in memory performance is unacceptable if the eventual costs are not consistent with what the marketplace can afford.

Figure 1 shows the present state-of-the-art in so-called conventional memory technology, in terms of capacity and access time. The technology ranges from the relatively small but fast semiconductor memory through moving head disc memory to the larger and slower bulk storage devices such as magnetic tape. Clearly most memory and storage technology is confined to magnetic phenomena. The exceptions to the magnetic dominance have been at the low-capacity, high-speed end with semiconductor technology and at the large capacity slow access end with the IBM 1360 and Precision Instrument Model 190 bit-by-bit optical technology.

The trends in memory and storage technology indicate a gradual (although sometimes rapid) trend up and to the left, i.e., toward larger and faster devices. Consequently, the aim of early holographic memory researchers was toward those areas where the payoff would be largest, i.e., 10^7-10^9 bit capacities with 1-10μsec access time for disc replacement. Similarly, the promise of extremely high data packing density afforded by holographic encoding encouraged research activity at the upper end of the spectrum to achieve terabit capacity.

Before we discuss the progress made to date, we shall review some of the basic concepts and components used in almost all holographic memory and storage devices. The holographic approach does not record individual bits but rather the optical interference pattern produced by two coherent light beams, one of which contains the information about the data to be recorded. This is shown in Figure 2. In general, the holographic approach requires that several unique components be arranged to produce a memory device. These are a source of coherent light, such as a laser, some rather conventional, but sophisticated, optical elements, a page composer to transduce electrical data into a form which can spatially modulate a light beam, a recording medium which records the information as a hologram, an array of detectors to transduce the reconstructed light pattern back into electrical signals and finally light deflectors or moving recording media to address various hologram positions on the recording media. The principal advantages of recording data in holographic form include: (1) a natural distributive encoding by recording the information over the entire hologram rather than at discrete points, thereby reducing susceptibility to dust, scratches and recording media imperfections, (2) the data reconstructed during readout is projected directly onto a fixed photodetector array with 10^3 to 10^5 bits appearing simultaneously (i.e., a page-oriented parallel access) and (3) insensitivity of the recording medium placement relative to the detector array.

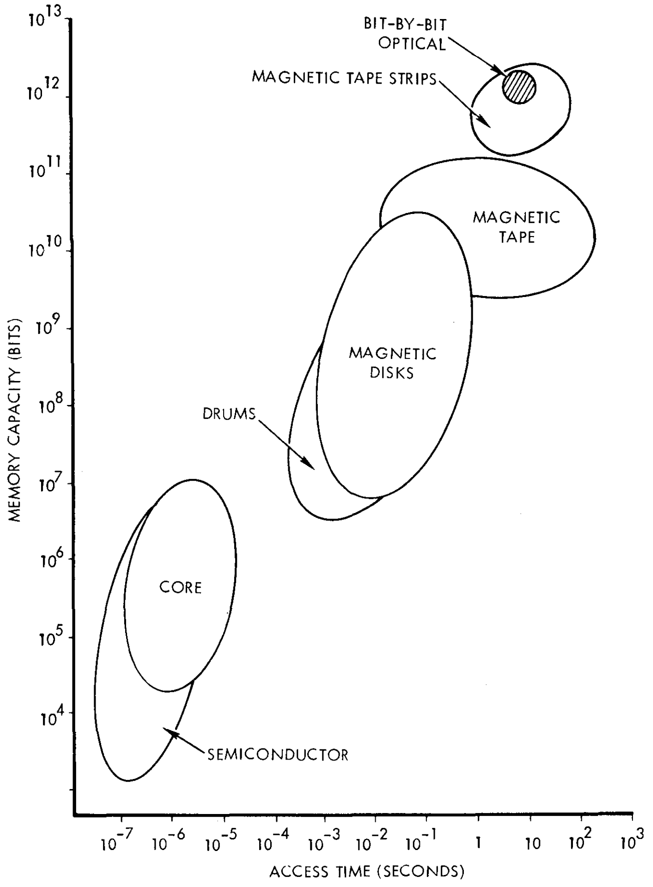


Figure 1—Current memory technology

A logical distinction between various holographic memories can be made by first considering those which are truly read/write memories and are aimed at existing mainframe and peripheral technology and those which are directed at mass storage applications. While the basic technology in both applications is somewhat similar, the approaches to solutions require emphasis on different components.

READ/WRITE HOLOGRAPHIC MEMORIES

The major effort to date on read/write holographic memories has addressed capacities between 10^6 and 10^9 bits with access times measured in microseconds. Several domestic companies, including Harris Corporation, Bell Labs, IBM, and RCA, as well as several foreign laboratories at Siemens, Nippon, Hitachi and Thomson-CSF, have either developed breadboard memories or are actively pursuing development of major components which are required in holographic memories. It is beyond the scope of this paper to discuss in detail the work of each laboratory and to dwell on the progress made in developing individual components. It is sufficient to say that although progress on each holographic memory component has been significant over the past decade, a truly viable cost competitive memory has not emerged from the breadboard stage into the marketplace. Because of the interactive nature of all holographic memory components, a major advance in one area may produce only minor improvement in system performance.

Perhaps the two components which have received the greatest attention are the input page composer and the holographic recording media. In both cases, the performance limitation has been dictated by the availability of suitable materials. The page composer should contain between 4000 and one million elements. Various materials have been or are being considered, including PLZT, liquid crystals, thin deformable membrane mirror arrays and cadmium sulfide. All currently suffer from one or more shortcomings including uniformity, speed, contrast ratio and stability. The holographic recording materials have also received considerable attention and all candidates fall short of the desired properties of high efficiency, high sensitivity and long lifetime. Significant progress has been made in developing beam deflectors and photodetector arrays, stimulated in part by the requirements imposed on these devices by holographic memory researchers. The performance of beam deflectors has nearly doubled in the past ten years and sophisticated multielement two dimensional photodetector arrays were essentially unavailable ten years ago. Research in all component areas is continuing. We can expect significant breakthroughs only by application of new materials or through better understanding and perfection of existing materials.

Until recently, heavy emphasis was placed on preserving high speed access with no moving parts while increasing capacity. Clearly the target was disc replacement. Analyses of the constraints imposed on common optical components such as lenses as well as a better appreciation of physical limitations imposed on the electro-optical components has led most investigators to revise their pre-

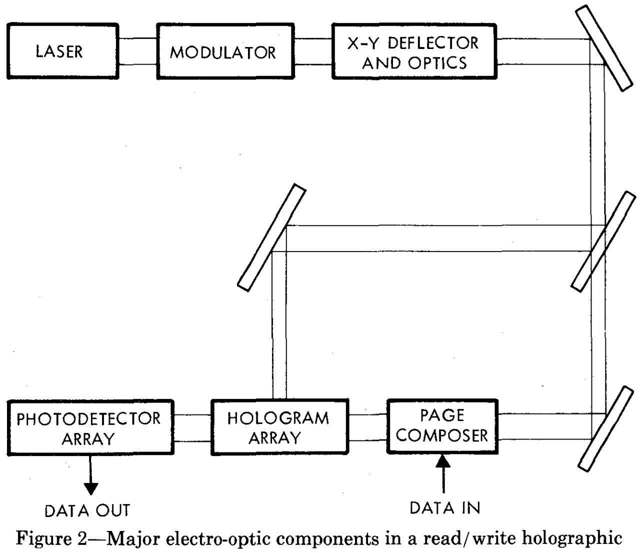


Figure 2—Major electro-optic components in a read/write holographic memory

dictions of ultimate read/write holographic memory performance, regardless of cost considerations. The capacity of nonmechanical, practical read/write holographic memory with microsecond access time is likely to fall between 10^8 and 10^9 bits, not up to 10^{12} bits as was believed earlier. In any case, because of costly electro-optical components, this type of memory will be characterized by high cost per bit. Only in highly specialized applications where technical performance plays an over-riding part will we see this memory used. Even so, it is not likely to emerge from the research laboratory before the end of this decade.

READ-ONLY MEMORIES

Read-only holographic memories typically use film as the recording media. Once exposed, the film record is removed from the recorder, developed by normal techniques, and placed in a holding area until data retrieval is required. If any portion of the recorded data must be changed or updated, the entire record must be re-recorded and replaced within the memory. Read-only memories, therefore, are best suited for archival, non-dynamic memory applications or applications where updating is relatively infrequent.

Other than the recording media, the holographic exposure and data readout processes are similar to those used in a read/write memory. Similar devices are required to implement a read-only memory as are required to implement a read/write memory; hence, read-only memories are therefore constrained by similar device limitations. Possibly the most critical device limitation in read-only memory implementation has been the page composer. Since a one dimensional (instead of a two dimensional page composer) is typically required, device capability in this area has recently been improved enough to allow system applications.

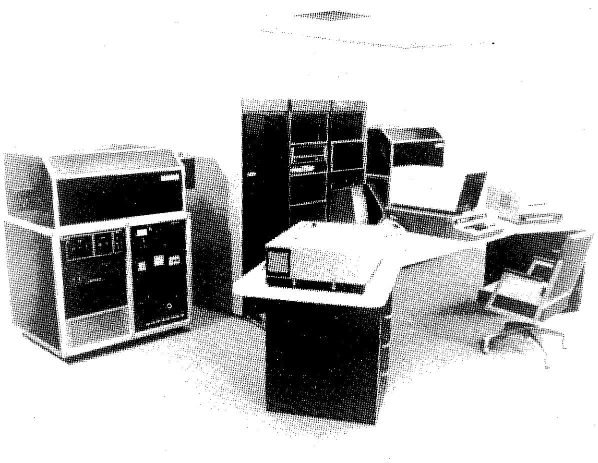


Figure 3—HRMR research prototype system

One way to overcome the page composer limitation in holographic recording is to use a synthetic hologram approach. In this approach, a film intensity function is calculated by a special purpose digital processor and scanned onto the film by a scanning device. The resulting film exposure has nearly the same reconstruction properties as does an interferometrically generated hologram.

The recent advances in materials and components have allowed production of a few prototype holographic memories. For example, let us discuss some specific hardware systems being developed by Harris Corporation, Electronic Systems Division.

Synthetic holography has been successfully applied to the recording and storage of digital data in the Human Read/Machine Read (HRMR) System developed by Harris Electronic Systems Division under contract with the Rome Air Development Center. A research prototype, shown in Figure 3, was delivered in May, 1973 and an engineering prototype is currently under development.

The HRMR System addresses the document storage, retrieval and dissemination problem which is impacting both government and industrial complexes having large document data bases. The HRMR concept is based upon annotating a standard microfiche with the digital equivalent of the associated images. Optical readout of the digital data directly from the microfiche facilitates storage, retrieval and dissemination of data to both local and remote locations.

A direct extension of the concept is the full utilization of the microfiche film chip for digital data recording. Thirty megabits of user data per film chip is presently being realized at a packing density exceeding one megabit per square inch. Since this packing density is significantly below theoretical limitations, considerable improvement can be anticipated as components and techniques are further refined.

Utilization of holography as the digital data recording technique in the HRMR System provides an inherent immunity to dust, scratches, and film imperfections associated with practical hardware which is capable of functioning in an operational environment. Only normal microfilm storage environmental conditions are required. The recorded data is archival and optical readout of data is nondestructive. This results in a virtually permanent record and contrasts with magnetic media which suffers from signal loss and deterioration due to readout and long term storage. Of further benefit, the positional invariance property of holography facilitates readout and allows relatively simple and economical hardware configurations.

Microfiche generation in the HRMR System is accomplished by means of a laser recorder which scans onto a film chip both the human readable images and the synthetically generated machine readable holograms containing digital data. Digital data is recorded sequentially onto the fiche in 500 kilobit blocks and at data rates compatible with magnetic tape drives. Fiche are automatically developed and all digital data is verified by means of parity bits which are appended during the recording process.

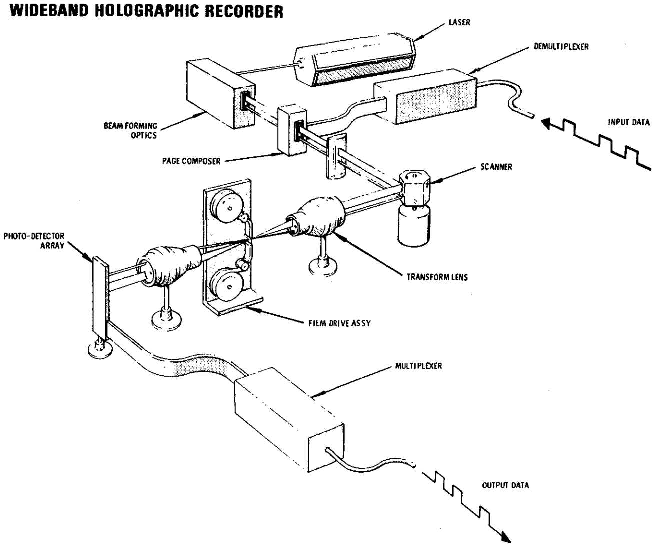


Figure 4—Wideband holographic recorder exploratory development model

Since the HRMR System storage media is oriented around the standard microfiche film chip, there is a maximum compatibility with commercially available microfiche handling equipment. It has been a straightforward development to configure a medium scale microfiche storage and retrieval device capable of handling approximately 7000 microfiche. Total digital store of this device is $2(10^{11})$ bits.

In the present HRMR System configuration, the mass memory is on-line to a PDP-11/45 computer. Random fetch of any 500 kilobit data block stored within the memory is provided with a maximum access time of less than 15 seconds. Transfer rates are compatible with DEC Unibus cycle times, but can be tailored to any host computer's channel characteristics and data absorption rates.

Because the holographic technique used in the HRMR mass memory is read-only, utilization of the system is projected to be oriented primarily toward archival data store applications in which the data placed in the memory is non-dynamic. Such data bases are quite common in both governmental and industrial organizations and are typically characterized by large magnetic tape libraries. A magnetic tape, having typical block sizes and utilizations, can be holographically recorded on one to two fiche. The 7000 fiche storage capacity of the holographic memory provides on-line access to approximately 3500 magnetic tapes with access times a fraction of manual retrieval, mount and read times. Based upon user requirements, additional holographic memory modules can be added to increase this capacity at least an order of magnitude.

Possibly the most significant characteristic associated with the HRMR System's holographic memory is simplicity of operation. The HRMR configuration has integrated into a mini-computer system, a $2(10^{11})$ bit memory and has made this data store available at a media cost of approximately 2.5×10^{-6} cents per bit. In contrast to many other conventional mass memory approaches, which record sequential blocks and must retrieve blocks sequentially, the HRMR approach provides random access to data blocks without a sacrifice in overall access time.

While the use of the synthetic holography for storage and retrieval of digital data on microfiche provides a solution to document-oriented mass memory requirements, different recording techniques and physical record formats are more suitable to other types of applications. For example, the storage and retrieval of digital data in very large data records at extremely fast recording and readout data rates can be best handled using roll film and interferometric holography.

As with magnetic tape recorders, the large supply of continuously moving recording media allows very large (i.e., tens to hundreds of megabits) data records to be recorded and played back with little interface buffering. Roll-film formats also allow sustained data processing at hundreds of megabits per second. Thus, holographic recording on roll-film offers an extension of the large data buffer capabilities now offered by high speed instrumentation-type magnetic tape recorders. Currently, single transport magnetic tape recorders can operate at recording and playback speeds of up to 80 or 90 Mb/s and can store data at a linear density of about 600Kb/inch on one-inch wide tape. In comparison, holographic techniques can be used to record and reproduce digital data on single transport devices at several hundred megabits per second at linear packing densities that are at least six times greater than now practical with magnetic tape.

The Wideband Holographic Recorder Exploratory Development Model, also developed by Harris Corporation, Electronic Systems Division under contract with the Rome Air Development Center, uses roll-film format and an interferometric approach. This system has demonstrated the recording of data at 400 Mb/s and the readout of data at 40 Mb/s—a 10:1 slow down. Using interferometric Fourier transform holography, data is recorded in one dimensional holograms spaced on 15 micron centers across the film. Over 1500 such holograms, each containing 128 data bits, are recorded across the film which is continuously moving at approximately 2 meters per second. Figure 4 is a functional diagram of the system showing both the recorder and reader components.

About $3(10^{11})$ bits of data can be recorded on a 5000 foot roll of 35 mm film. Using recording rates of up to 500 Mb/s, non-stop recording could be sustained for a period of up to 9.5 minutes. Readout rates reduced by 10:1 or 100:1 from record rates are easily implemented and current development activities promise that readout speeds equivalent to record rates will soon be possible. Although recording is readily accomplished using roll-film supplies, once recorded, the film can be segmented and cassette mounted when faster, more random data access or distribution of duplicate data packs is desired.

In some applications, both the automatic storage and retrieval features offered by the microfiche recording format and the higher speed recording and readout capability provided by roll-film formats are desirable. Under contract to NASA Marshall Space Flight Center, Harris Electronic Systems Division is developing a Holographic Memory that incorporates some of the features of both HRMR and the Wideband Holographic Recorder. One of the goals of this system development is to record up to 80 Mb of data on a microfiche which is formatted into randomly addressable files.

CONCLUSIONS

During the past several years we have witnessed a considerable effort which was and still is being undertaken by many research laboratories to apply the principles of holography to a broad spectrum of memory and storage applications. The research is directed toward both the intermediate-capacity, fast access time read/write memory market as well as the large-capacity, longer access time read-only storage devices. Effort is also continuing in the specialized area of ultra-high speed transfer of data into and out of large intermediary bulk stores.

Read/write memories have not yet emerged from the research laboratory into the commercial marketplace. Efforts have been hampered primarily by the unavailability of suitable materials which are needed to configure several key memory components. Even if we assume that material and other technological problems are overcome, the prospects that holographic memories will seriously challenge other existing and emerging technologies before the end of this decade, indeed if ever, is unlikely. The inertia of magnetic technology coupled with remarkable yearly improvements in packing density, access time and transfer rates presents a formidable challenge to those who desire to penetrate that particular segment of the market.

The prospects for read-only holographic memories which have multi-microsecond access time and 10^8 to 10^9 bit capacity appear to be better because problems associated with high-speed page composers and reusable storage media are obviated. Unfortunately, the read-only property will limit its usefulness to special applications where data volatility, extreme environments and data security overshadow cost considerations.

The prospects for application of holographic techniques to read-only bulk storage appear to be much better. Systems with capacity between 10^{11} and 10^{13} bits and with multi-second access time are currently being built as engineering developmental units. At these capacities, costly electro-optical components can be justified. On a more modest scale, 10^7 bit capacity, 1.5 second access time read-only storage units are already commercially available for application to the point-of-sale credit card verification problem.

To date, archival read-only optical memories using bit-by-bit recording techniques have penetrated a portion of the large capacity storage market. This market will grow as demand for larger common data bases increases. Although today's market penetration is being made by magnetic and bit-by-bit optical techniques, holographic techniques offer the most promising cost-effective approach for achieving high transfer speeds and large capacity. We believe that holographic storage devices will become commercially attractive well before the end of the decade.

REFERENCE

1. Leith, E. N. and J. Upatnieks, *J. Opt. Soc. Am., 52,* p. 1123, 1962.

BEAMOS—A new electronic digital memory

by W. C. HUGHES, C. Q. LEMMOND, H. G. PARKS, G. W. ELLIS, G. E. POSSIN, and R. H. WILSON

General Electric Company
Schenectady, New York

INTRODUCTION

BEAMOS, for **B**eam **A**ddressed **M**etal **O**xide **S**emiconductor, is a new technology for auxiliary memories based on an electron beam which reads and writes data on a simple unstructured MOS chip. It can store data for months with or without power.

This memory combines large data capacity, high data transfer rate and a highly flexible data format. BEAMOS use in data systems should result in significant savings in both hardware and software costs.

This technology also makes possible auxiliary memories with the ruggedness and non-volatility needed for military systems.

Static auxiliary memories such as cores or semiconductors become extremely expensive and decline in overall reliability when large amounts of storage capacity are required. This results in severe constraints for the designer of military information systems. BEAMOS memories provide an attractive alternative.

DESCRIPTION OF THE BEAMOS MEMORY

A complete BEAMOS memory consists of one or more BEAMOS modules, address and interface logic, control circuits, and power supplies.

A 32 million bit capacity was chosen for initial development because it was the most viable size for the applications envisioned. Lower capacity modules are relatively easy to design and build but have a higher cost per bit. Higher capacity modules provide less modularity in systems design, but on the other hand, have lower per bit component cost.

Physical description

The BEAMOS module, Figure 1, contains a memory plane and electron beam accessing system enclosed in a sealed, evacuated envelope. In military systems the module is shock mounted in a rugged outer shell.

Memory plane

Information is stored on the memory plane,[1] Figure 4, which consists of four BEAMOS chips mounted on a baseplate. The essential structure is shown in Figure 2. It consists of a film of aluminum evaporated on a thin insulating layer, such as silicon dioxide, formed on a silicon diode. All layers are arranged in continuous, unstructured planes. The n and p type silicon layers are connected electrically to form a back-biased diode. The operating mode of the memory is controlled by the application of a voltage across the oxide layer. A positive 40 volt bias is applied to write and a negative 40 volts to erase. A read bias of zero volts allows many reads before rewriting. Destructive readout can be obtained with a negative 40 volt bias.

Electron beam addressing

The conventional way to address an electron beam to a given location in a field is through a single stage of deflection. The maximum number of addressable sites is determined by the accuracies of the deflection structure and the driving amplifier, and by the stability of the power supplies. A practical open-loop limit is approximately 1000 addressable sites along a line, and hence a million addressable spots in a square field. In the BEAMOS module, addressing of the electron beam to the memory sites is achieved in a special electron optical structure. This employs two stages of deflection, in combination with a unique matrix of small lenses. This array of lens/deflection systems, called the Matrix Lens,[2] permits addressing a very large number of discrete memory sites with a single electron beam.

The complete electron optical system of the BEAMOS module is shown in Figure 3. The first digits of the address are applied to a digital-to-analog converter. This converter generates an analog voltage which is applied to the first deflection stage called a lenslet selector. This directs the electron beam into one of the small lenslets. A lenslet is simply a pair of aligned holes in two metal plates. A voltage applied between the plates produces an electron lens which focuses the beam to an extremely fine spot. In addition to its focusing capability, each lenslet has an individual deflection structure, integral with the matrix lens.

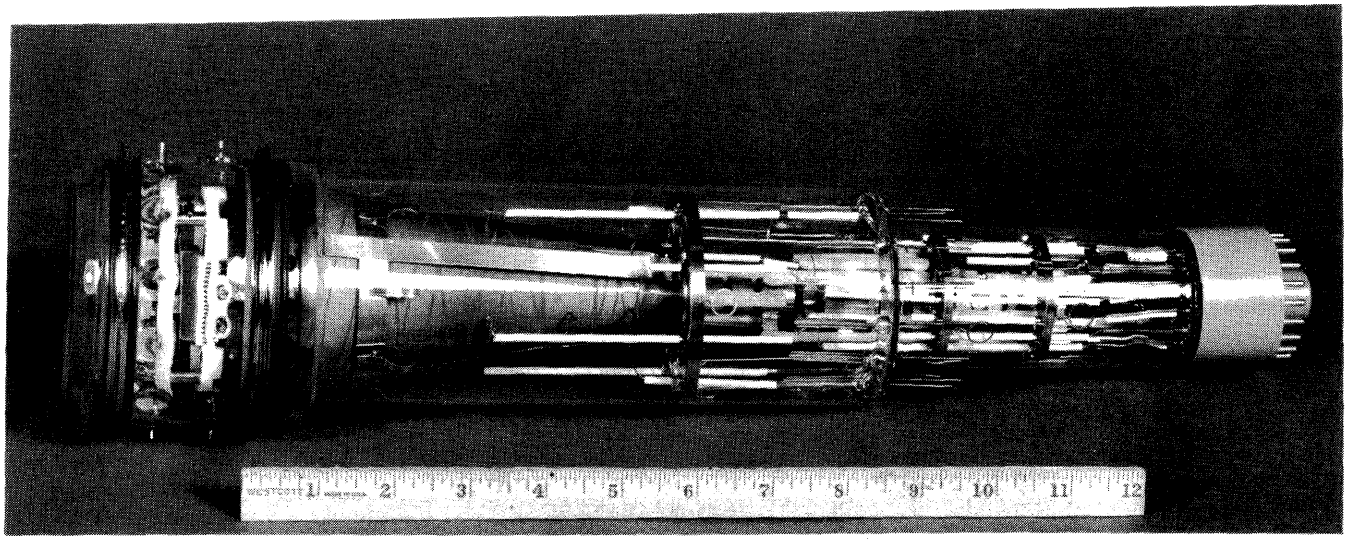


Figure 1—BEAMOS memory module

The second part of the digital address is applied to another digital-to-analog converter, producing a deflection voltage that is applied to this second deflection structure called a page selector. This structure directs the beam into the desired memory site within the lenslet field. The two deflection stages are effectively independent. Minor beam positioning variations in the first stage do not cause significant errors in the position of the bit site addressed. Figure 4 is a photograph of a Matrix Lens together with the memory plane. The grid structure is the page selector matrix.

Operating sequence

To record data in the module the following sequence is performed. First the address and operating mode are selected by simultaneously entering the lens and page addresses and switching in the desired oxide bias. This moves the beam to the beginning of the page. From this point the beam is stepped across the lenslet field one bit site at a time while the electron beam is synchronously turned on or off as required by the input data. An output signal is available from a read amplifier which can be monitored as a partial check on the write operation.

The sequence is the same for the other operations (read, destructive read, and erase), except that the electron beam is turned on at every bit site.

Signal life

The BEAMOS target stores data for a long time but not permanently. Thus it is necessary to rewrite the files occasionally whether they are used or not. This function can be carried out by reading the data into a block buffer and rewriting. Signal decay is very slow with the power off or with power on and zero or negative oxide bias values. With

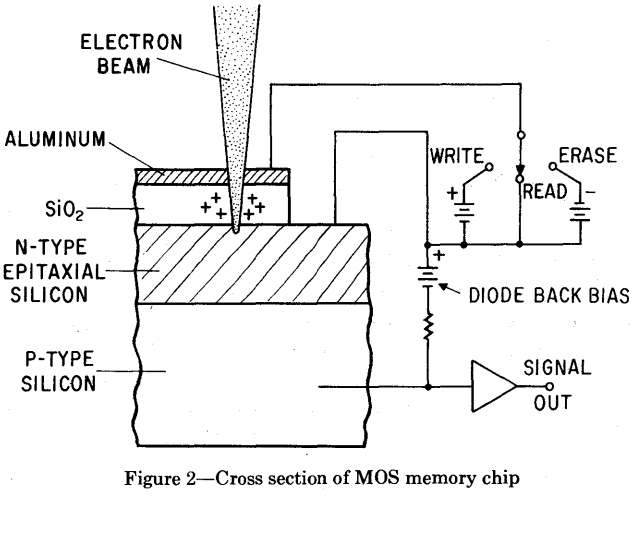


Figure 2—Cross section of MOS memory chip

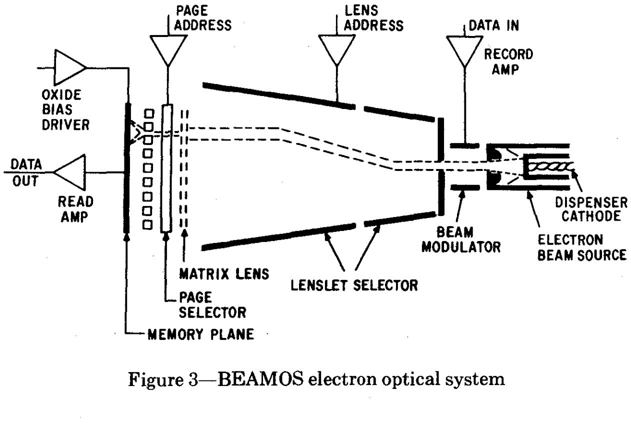


Figure 3—BEAMOS electron optical system

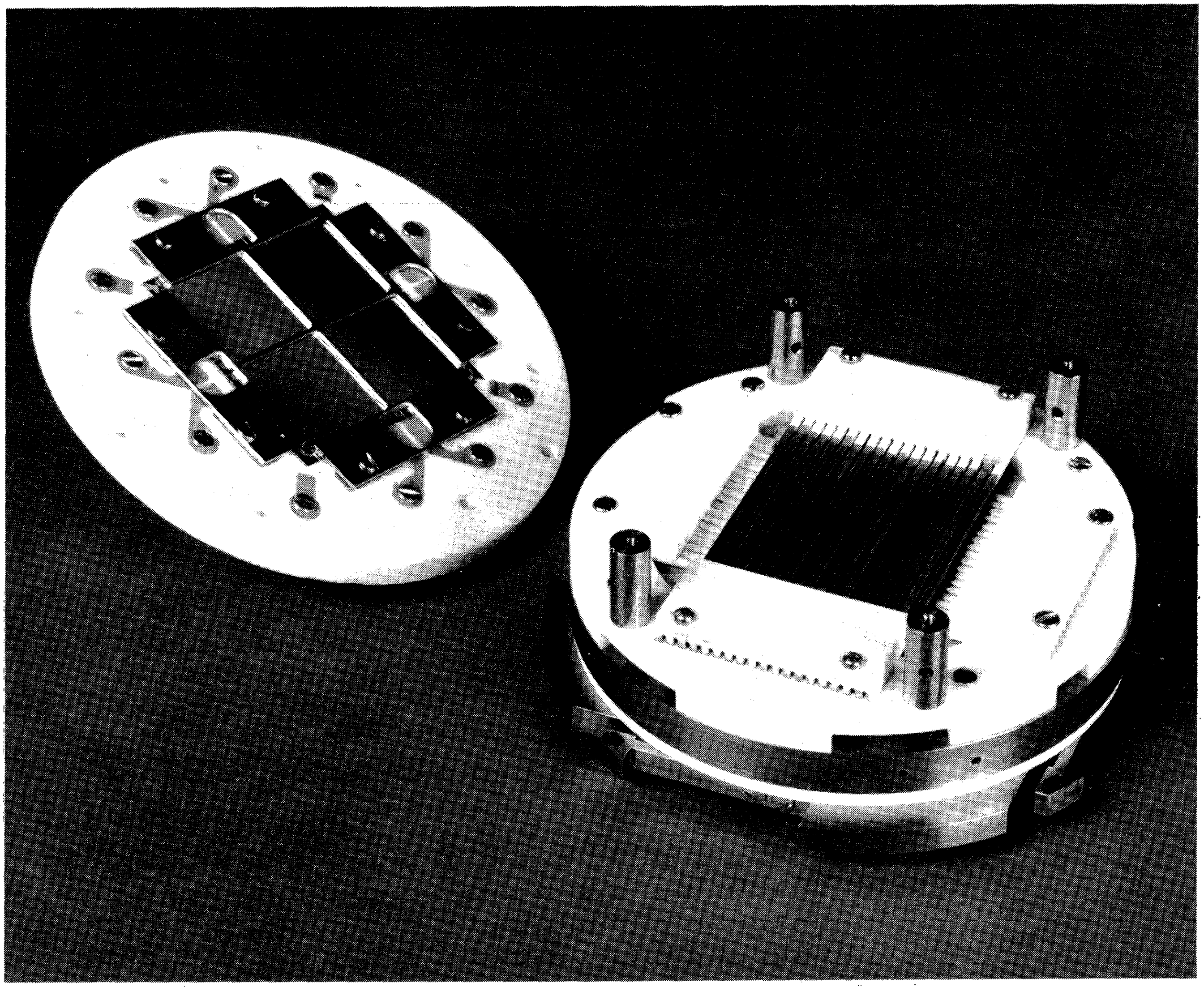


Figure 4—Matrix lens showing page selector and memory plane

power off, the measured signal is 90 percent of its initial value after five day storage and about 80 percent after one month. Decay is somewhat faster at positive bias. The decay characteristic is only slightly temperature dependent with variation of no more than +10 percent in signal level between −40 and +70°C.

The output signal from the module is also reduced slightly by each readout. After approximately 20 reads it must be rewritten. It is advantageous to operate the module such that several reads are always possible, so that re-read cycles can be used as a part of the error recovery strategy. Data refresh can be done after each read or when necessary, based on the readout signal level.

MEMORY SYSTEM CHARACTERISTICS

The BEAMOS module can be configured into memory systems which have a wide spectrum of characteristics. This design freedom results from the non-structured nature of the storage plane and the flexibility of electron beam addressing.

Data format

The data layout used in the BEAMOS module can be selected at the discretion of the systems designer. The page length can vary over wide limits, however, a guardband must be left between pages to assure that activity on one page does not affect neighboring pages. A guardband is also required between lenslets. There are no built-in restrictions on word length and redundancy can be included within the page as required for error control.

An example of a particular format based on a 8448 bit page is shown in Figure 5. Here each page occupies 24 data lines extending completely across the lenslet field. A

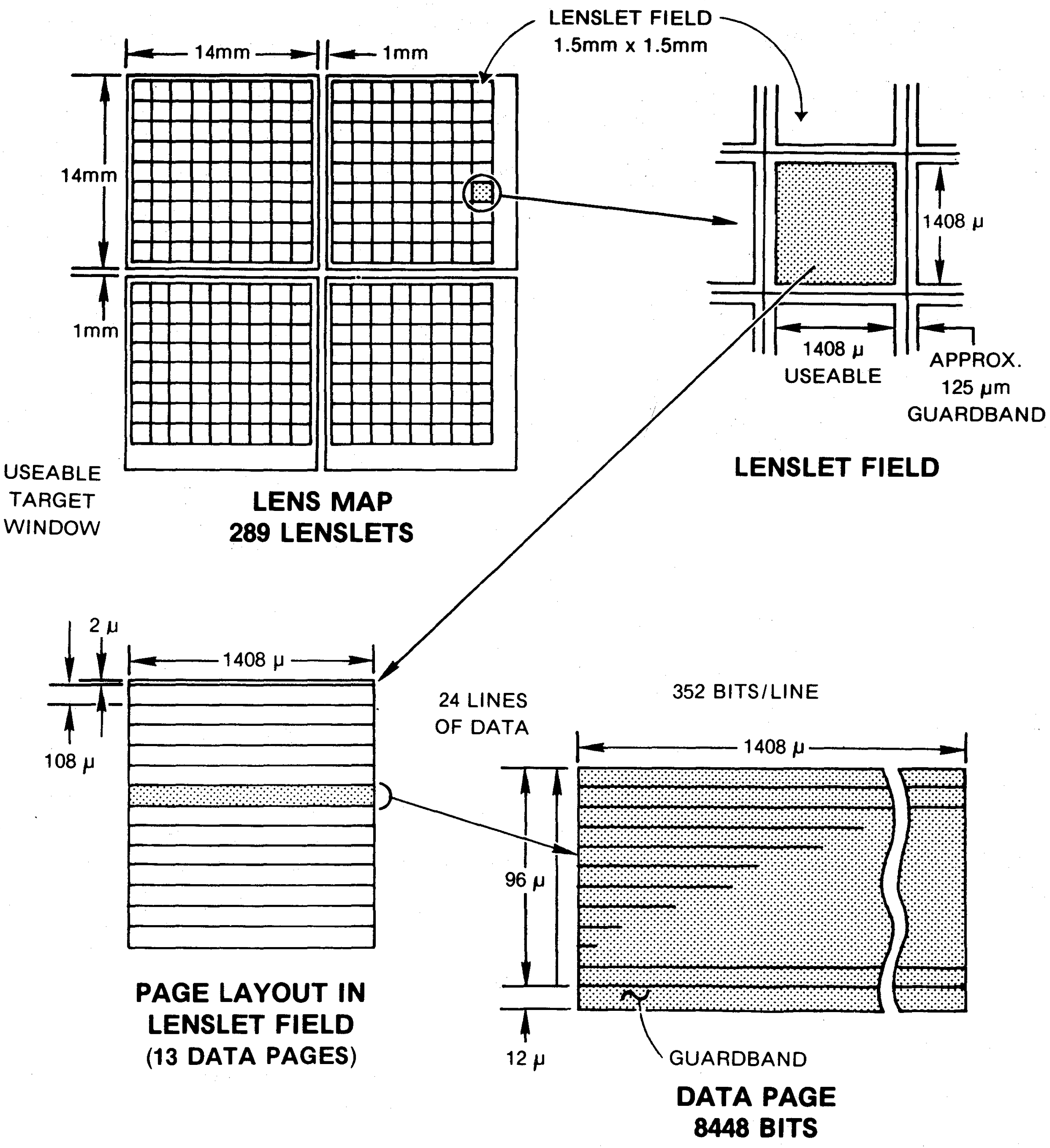


Figure 5—Data format

12 micron guardband is provided between pages and a 125 micron guardband between lenslets.

System configuration

Systems needing 32 million bits capacity require only a single module. In larger systems, multiple modules are used and advantage can be taken of the fact that much of the electronics can be shared among several modules, thus minimizing the cost per bit of the system. Figure 6 is a block diagram of a multimodule system in which the modules are addressed one at a time. In this configuration, the modules share the lens and page select amplifier, oxide bias circuitry, most of the digital electronics and the power supplies. This provides minimum per bit cost and maximum reliability with page access times in the 30 microsecond range and data rates up to 10 megabits/second. Data capacity for a 20 tube system would be at least 600 million bits.

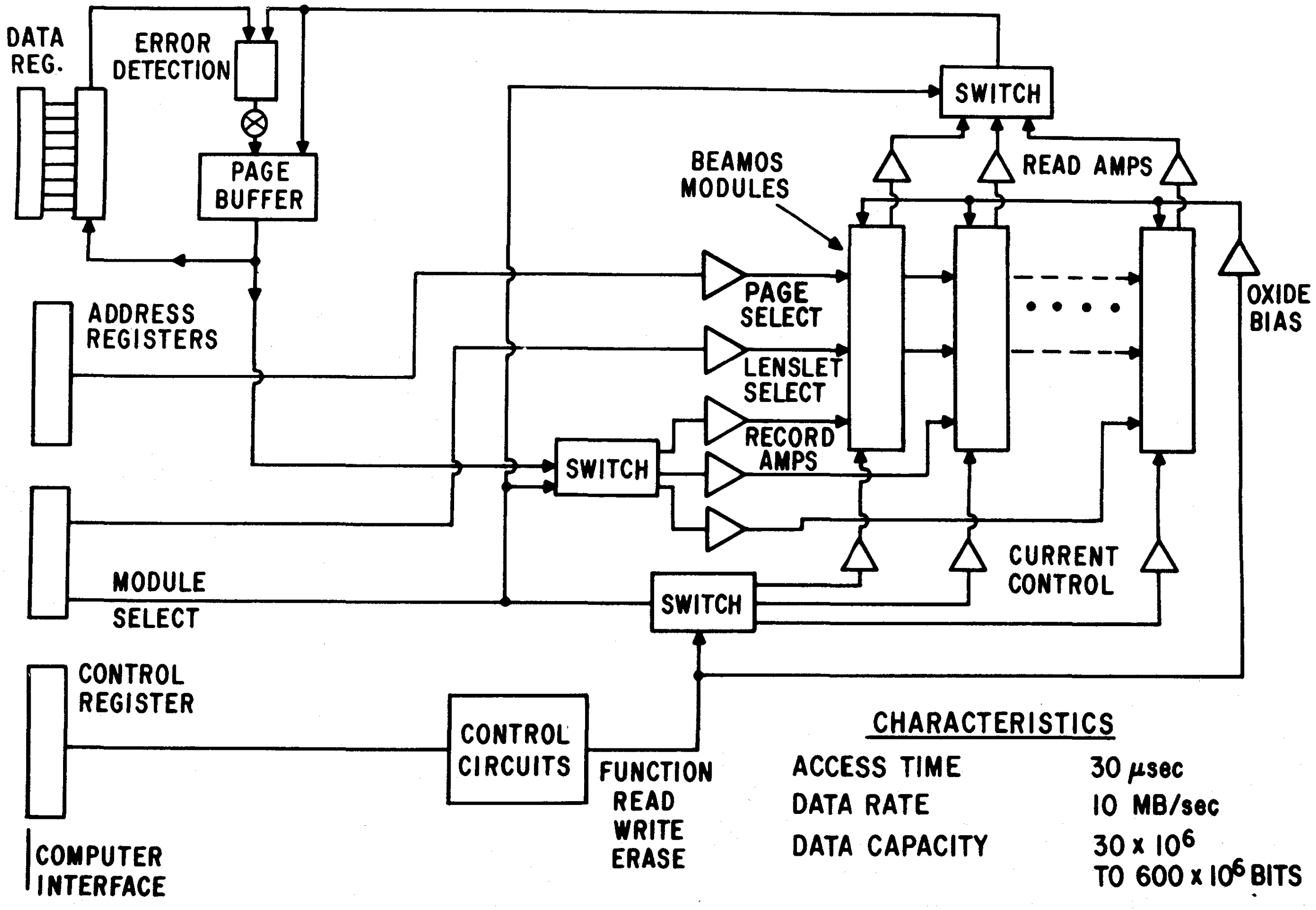


Figure 6—Multimodule system—serial operation

In some systems, much higher data rates are desirable to minimize service time. This can be provided by accessing modules in parallel. That is, all, or several, modules are simultaneously addressed to the same location and each writes or reads a part of the total page. Again much of the electronics is shared among the modules but since more digital electronics is required the cost per bit can be expected to go up somewhat.

This type of system can have a 30 microsecond access time, 600 million bit capacity and data transfer rates up to 200 megabits/sec.

Access time

Access time, the time required to move the electron beam to the beginning of a page, is fundamentally limited by the transit time of the beam through the optical system, about 30 nanoseconds. The practical electron beam access time limit is imposed by the switching and settling time of the deflection amplifier driving the capacitance of the all electrostatic lenslet and page selectors. This can be in the range of a few microseconds.

The time required to change the bias of the oxide when switching from one operation to another (i.e., read to write) requires a longer time because the relatively large oxide capacitance must be charged through the resistance of the n-layer in the memory plane. This can be accomplished, with time for the disturbance caused in the readout amplifier to die out, in about 30 microseconds.

Thus, the access time can be a few microseconds for successive calls of the same operation and up to 30 microseconds if an operation change is involved.

Data rate

The recording data rate is determined by the beam current available. In the present modules this is 10 megabits/sec. Considerably less beam current is required to read, making it possible to read at higher rates. Target frequency response is not a limiting factor. At higher read rates fewer reads can be made before the data has to be rewritten because more beam current is required. At 10 megabits/sec about 20 reads are possible, while at 100 megabits this would be reduced to six.

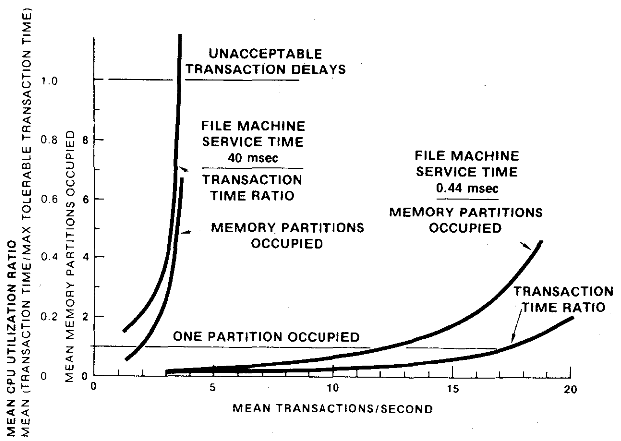


Figure 7—System performance comparison

Service time and some system consequences

A meaningful number describing the speed of a memory system is *service time*. This is the average time between the acceptance by the memory of a new command and the earliest time the next command can be accepted. In many applications the service time of the file equipment can have a dramatic effect on the performance of the total system. As an example of this, consider a transaction processing system in which a data base is shared among many users. The system has a CPU and a working memory with space for the file management software and eight partitions for transaction programs.

Requests for CPU and memory time come in on a random basis from the users and the required programs are brought in from the file and assigned a portion of the main memory. The programs are executed in turn until they are completed or an operation which requires considerable time is encountered, for example, an input/output command or a memory access. When this occurs the operation is initiated and the CPU goes on to the next transaction. When transactions begin to come in at a rate approaching the average processing rate, queues build up and the waiting time may become unacceptably long or the number of transaction partitions exceeded.

The software for such a system must be very complex to cope with all eventualities. A particularly troublesome situation occurs when two transactions which are resident in main memory call for access to the same file; one to modify it and the other to read it. In this case it may be necessary to roll both transactions back to their initiation to decide whether the modification or read should occur first.

This kind of system has been studied by computer simulation to determine the effect of service time on its performance.[3] The results are dependent upon the capabilities of the CPU and file memory and upon the nature of the transactions in more detail than can be discussed here, but Figure 7 illustrates what can happen. These calculations were made for a real system operating on a transaction mix typical of those occurring in a manufacturing plant automation data system.

With a file service time of 40 milliseconds which is typical of disk devices the operation of the system is limited by the file memory. Only three to four transactions/second can be accommodated before unacceptable delays begin to occur. At this point all of the memory partitions are occupied with transactions in process, greatly increasing the chances for conflicts in the use of the data base.

If the service time is decreased to 0.44 milliseconds, as would be possible with a BEAMOS memory, the capacity of the system is increased to the point where over 20 transactions/sec can be processed. The transaction rate would be three times that achievable with the slower file with only one memory partition occupied. This means that transactions could be processed serially in order of arrival, with a consequent simplification in data base protection and operating system software and with less main memory.

SUITABILITY FOR MILITARY APPLICATION

A version of the BEAMOS memory is presently under development for U. S. Army ECOM and assessments of its suitability for military environments look very promising.

Military environment

A major advantage of BEAMOS for use in a military environment is that it is all electronic, has no moving parts and is completely sealed. No parts are especially sensitive to vibration and the most important section in controlling beam position, the matrix lens/target assembly, is very rugged. Tests of this component suggest that it can easily be applied in typical aircraft vibration environments. By using familiar isolation techniques, application can be extended to very severe environments. Tests of the BEAMOS storage plane over the range $-50°C$ to $+70°C$ indicate that charge storage is not greatly affected by temperature and while there are some changes in target characteristics, it is possible to design the system to accommodate them. The matrix lens/target assembly is made of materials with matching coefficients of expansion so that variation from -40 to $+70°C$ should produce less than 0.1 micron variation in the position of the addressing beam.

The memory module requires a magnetic shield to protect against external fields, but a shield weighing about 10 pounds will accommodate fields up to 10 gauss. Considerably lighter shielding will suffice in less severe environments.

The BEAMOS memory plane is especially tolerant of ionizing radiation since it depends on high radiation levels for its operation. Ionizing radiation equivalent to 10^5 Rads (Si) will cause no more than 10 percent reduction in signal level. No permanent radiation damage has been observed up to 6×10^7 Rads. Neutrons will generate permanent damage but 10^{14} neutrons/cm^2 are required to cause significant reduction in signal level.

Military packaging

An important consideration in most military equipment is physical size and power consumption. A packaging study has shown that a single module BEAMOS memory can be packaged in a volume of 2-3 cubic feet and will require about 250 watts of power. The study assumed packaging techniques similar to those used in aircraft.

Reliability

The mean operating time between failures is a part of most military equipment specifications. It is useful to make an estimate of what might be expected in a memory of this type. A single module system was considered and by making a component count and applying component failure rate data from the Military Standardization Handbook[4,5] the following MTBF figures were determined.

Digital Circuits	30,000 hours
Analog Circuitry	36,000 hours
Power Supplies	20,000 hours
Resultant MTBF of Electronic Circuits	9,000 hours

It was assumed that the module itself would be replaced on a preventive maintenance schedule. The electron source[6] was a barium dispenser cathode.[7] At the cathode loading employed the dispenser cathode used in these devices can be expected to show only 10 percent drop in emission in 40,000 hours of use.[8] A 20,000 hour replacement period was selected, and assuming random failures of 20 percent prior to replacement, the combined MTBF for the electronics and the BEAMOS module is calculated to be 8,000 hours. This compares very favorably with other memories of this size and would be even more favorable for multimodule systems.

DEVELOPMENT STATUS

The 32 million bit BEAMOS memory module is being built in pilot quantities. These modules are being tested and evaluated in a computer controlled test system. Modules have been operated at a data density of 40 million bits per square inch which produces a module capacity of 32 million bits. They have been tested at data rates of 10 megabits/second and access times of 30 microseconds. Data storage time between refresh cycles can be as long as 120 hours.

MEMORY COST COMPARISONS

Estimates of the cost of a BEAMOS memory including the analog and digital electronics have been made and they compare favorably with competing technologies. The systems price will vary depending upon the number of modules used but is expected to be in the 0.02 cent per bit range for the present module size and density. The price can be expected to decline with future developments into the 0.001 cent per bit range.

POTENTIAL FOR IMPROVED PERFORMANCE

The BEAMOS concept has considerable potential for improvement with continued development. Operation of the memory plane with the matrix lens optics has been achieved in an experimental laboratory configuration at 100 million bits/inch^2.

Improvements in the electron optics can also increase the addressable target area in a module. A theoretical study supported by experimental evidence has demonstrated that 10 square inches of target space can be addressed at 160 million bits/inch^2. This provides a data capacity of over 1×10^9 bits per module.

Theoretical studies of the density and speed potential of the target predict that operation should be possible at 400×10^6 bits/in^2 (1.3μ bit spacing) and data rates of 10 megabits/second.

ACKNOWLEDGMENTS

Partial support for this development has been provided by the U.S. Army Electronics Command. The authors wish to thank J. F. Norton, T. H. Klotz, H. W. Grup, W. R. Bozony, and W. D. Barber for assistance in memory plane and module testing; E. C. Buschmann, J. F. Bedard, J. K. Fisher, and R. A. Wagner for their contributions to module fabrication; J. S. Rosczak and D. P. Lalla for assistance in memory plane processing; R. C. Raymond for the system modeling; and P. E. Pashler, R. W. Redington, H. L. Lester, R. O. McCary, R. H. Kerr, R. A. Sigsbee, C. A. Neugebauer and J. F. Burgess for technical guidance and advice. The contributions of J. C. Davis, M. Goldberg, R. J. Lucas, I. Reingold and P. F. Krzyzkowski of the Department of Defense are also gratefully acknowledged.

REFERENCES

1. Ellis, G. W., G. E. Possin and R. H. Wilson, "Diode Detection of Information Stored in Electron-Beam-Addressed MOS Structure," *Applied Physics Letters 24*, p. 419, 1974.
2. Lemmond, C. Q., E. C. Buschmann, T. H. Klotz and G. M. White,

"Electron Fly's Eye Lens Artwork Camera," *IEEE Transactions on Electron Devices, ED-21,* 9, Sept. 1974, p. 598.

3. Raymond, R. C., private communication.
4. *Reliability Stress and Failure Rate Data for Electronic Equipment,* Mil-HDBK 217B, 20 September 1974.
5. *Reliability Handbook,* RADC-TR-67-108.
6. Hughes, W. C., "Long Life-High Brightness Services for Demountable Guns," *Record of the Tenth Symposium on Electron, Ion, and Laser Beam Technology,* San Francisco Press, San Francisco, 1969.
7. Levi, R., "Improved Impregnated Cathode," *J. Appl. Phys.,* 26, 1955, p. 639.
8. Van Stratum, A. J. A. and P. N. Kuin, "Tracer Study on the Decrease of Emission Density of Osmium Coated Impregnated Cathodes," *Journal of Applied Phys.,* Vol. 42, No. 11, October 1971, p. 4436.

Area Director:
Ugo Gagliardi
Honeywell Information Systems Inc.
Waltham, Massachusetts

Interaction of technology and system architecture

Major hardware technology advances have and will have significant impact on system architecture by shifting feasibility constraints and causing new optimum structures to emerge.

This area develops the thesis that a number of very significant hardware technology advances are occurring or are about to occur. This, in turn, will result in significant changes in the: structure of software systems, productivity of systems, ease of use, operational practices, and range of applications of general purpose computing systems.

Some of the specific architectural areas considered are: multi-processors structures, storage hierarchy structure and control, firmware primitives for data-base applications, software architecture for memory and data management.

Sessions:

Professor Gerald Estrin, UCLA
Chairman—Introductory Panel Session

This introductory session will consist of two tutorial presentations pointing to the two following sessions. The first presentation will discuss major cost/density trends in main, disk and archival memories and how they are affecting system architecture and operating concepts for files, data bases and library systems. The second presentation will add major technology trends in processors and inter-connections and will discuss forms of Processor-Memory-Switch Architectures suggested by those trends.

Professor Stuart Madnick, MIT
Chairman—Paper Session—Processor Memory Switch (PMS) Architecture

Significant cost reduction advances in processor technology have now made multiple processor architectures economically feasible. In this session three specific examples are presented. The first uses an ensemble of up to 13 identical processors to implement a highly-reliable communications switching node (ARPA IMP). The second addresses the problem of asymmetric task scheduling in a multiprocessor system with heterogeneous processors (HITAC 8700's and 8800's). The third describes a highly modular network of microprocessors connected together by a common ring-bus.

Ornstein, S. M., et. al., Bolt Beranek and Newman, Inc., "Pluribus—A reliable multiprocessor"
Noguchi, Kenichiro, et. al., Hitachi, Ltd., "Design considerations for a heterogeneous tightly-coupled multiprocessor system"
Toong, Hoo-min D., MIT, "Microprocessor-based multiprocessor ring structured network"

Dr. Jeffrey Buzen, Honeywell Information Systems, Harvard
Chairman—Paper Session—Data and Memory Management Architectures

The hardware capabilities provided by new technologies and the software requirements generated by new applications are exerting a powerful influence on memory subsystem architecture. This session will consider both hardware and software factors and will include discussions of the conceptual and architectural support of data base systems, the use of LSI technology to provide logical processing capabilities within a memory subsystem, and the use of microprocessors to support both memory subsystem hierarchies and software hierarchies in a uniform manner.

Bachman, Charles W., Honeywell Information Systems "Trends in data base management—1975"
Glanz, Z. H., Thompson, P. M., University of Ottawa, "A data sorting system using high speed bus"
Madnick, Stuart E., MIT Sloan School, "Infoplex—A functional decomposition of large information management systems into a hierarchical microprocessor complex"

Mr. Richard P. Case, IBM
Chairman—Discussion Panel on Significance

The panel will consist of the three session chairmen plus one or two guests.

Pluribus—A reliable multiprocessor*

by S. M. ORNSTEIN, W. R. CROWTHER, M. F. KRALEY, R. D. BRESSLER, A. MICHEL and F. E. HEART
Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

INTRODUCTION

As computer technology has evolved, system architects have continually sought new ways to exploit the decreasing costs of system components. One approach has been to pull together collections of units into multiprocessor systems.[1] Usually the objectives have been to gain increased operating power through parallelism and/or to gain increased system reliability through redundancy.

In 1972, our group at Bolt Beranek and Newman started to design a new machine for use as a switching node (IMP) in the ARPA Network.[2,3] The machine was to be capable of high bandwidth, in order to handle the 1.5-megabaud data circuits which were then planned for the network. It was to have a high fanout to Host computers connected at a node. It was to come in all sizes (of processing power, memory, I/O) so that one could configure an individual IMP to meet the requirements of its particular location in the network, and change that configuration easily should the requirements change. Most of all, it was to be reliable.

The family of machines we have produced which meets these goals has been named the Pluribus line. The machines are highly modular at several levels and have a minicomputer/multiprocessor architecture. Although the largest configuration we have put together so far contains only 13 processors, we believe there are no inherent problems with considerably larger systems. The structure and details of some of the hardware are described in earlier papers.[4,5] Familiarity with these papers will be helpful in understanding the present paper, which focuses on the issue of reliability. We believe that reliability will become an increasingly common concern as multiprocessors become more commonplace, and we believe that we have gained some interesting insights into the solution of this problem.

THE MULTIPROCESSOR ARCHITECTURE

A novel feature of our design is the consistent treatment of all processors as equal units, both in the hardware and in the software. There is no specialization of processors for particular system functions, and no assignment of priority among the processors, such as designating one as master. We chose to distribute among the processors not only the application job (the IMP job) but also the multiprocessor control and reliability jobs, treating all jobs uniformly. We view the processors as a resource used to advance our algorithm; the identity of the processor performing a particular task is of no importance. Programs are written as for a single processor except that the algorithm includes interlocks necessary to insure multiprocessor sequentiality when required. The software of our machine consists of a single conventional program run by all processors. Each processor has its own local copy of about one quarter of this program and the remaining three quarters is in commonly accessible memory.

Hardware structure

Reliability was a main concern in planning the hardware architecture. Although we tried to build the individual pieces solidly, our main goal was to provide hardware which could be exploited by the program to survive the failure of any individual component.

The hardware consists of busses joined together by special bus couplers which allow units on one bus to access those on another. Each bus, together with its own power supply and cooling, is mounted on its own modular unit, permitting flexible variation in the size and structure of systems. There are processor busses each of which contains two processors, each in turn with its own local 4K memory which stores frequently run and recovery-related code. There are memory busses to house the segments of a large memory common to all the processors. Finally, there are I/O busses which house device controllers as well as certain central resources such as system clocks and special (priority-ordered) task disbursers which replace the traditional priority interrupt system. About half of the machine consists of standard parts from the Lockheed SUE line; the remainder is of special design.

As emphasized in our initial paper,[4] we were fortunate to have a very specific job in mind as we designed the system. This enabled us to place specific bounds on the problems we sought to solve. For example, the proposed initial setting within a communications network means that outside entities (neighboring communications processors, Hosts, users, etc.) may help to notice that

* This work was supported by the Advanced Research Projects Agency (ARPA) under contracts DAHC15-69-C-0179 and F08606-73-C-0027.

things are going wrong. It also means that recovery assistance is potentially available from the Network Control Center (NCC) through the network.[6,7] The system is designed generally to avoid reliance upon external help, but upon occasion such help is useful and therefore we have provided methods for permitting the system to be forcibly reloaded and restarted via the network.

Software structure

The problem of building a packet-switching store-and-forward communications processor (the IMP) lends itself especially well to parallel solution since packets of data can be treated independently of one another. Other functions, such as routing computations, can also be performed in parallel.

The program is first divided into small pieces, called *strips,* each of which handles a particular aspect of the job. When a task needs to be performed, the name (number) of the appropriate strip is put on a queue of tasks to be run. Each processor, when it is not running a strip, repeatedly checks this queue. When a strip number appears on the queue, the next available processor will take it off the queue and execute the corresponding strip. We try to break the program into strips in such a way that a minimum of context saving is necessary.

The number assigned to each strip reflects the priority of the task it performs. When a processor checks the task queue, it takes the highest priority waiting job. Since all processors access this queue frequently, contention for it is very high. We therefore built a hardware device called the Pseudo Interrupt Device (PID) which serves as a task queue. A single instruction allows the highest priority task to be fetched and removed from the queue. Another instruction allows a new task to be put onto the queue. All contention is arbitrated by standard bus logic hardware.

The length of strips is governed by how long priority tasks can wait if all the processors are busy. The worst case arises when all processors have just begun the longest strip. In the IMP application, the most urgent tasks can afford to wait a maximum of 400 microseconds. Therefore, strips must not generally be longer than that.

An inherent part of multiprocessor operation is the locking of critical resources to enforce sequentiality when necessary.[8] A load-and-clear operation provides our primitive locking facility. To avoid deadlocks, we priority-order our resources and arrange that the software not lock one resource when it has already locked another of lower or equal priority.

Status

During the early spring of 1974 a prototype 13-processor system was constructed. As this paper is being written (in the fall of 1974) two production copies have been constructed and are running. Each contains 13 processors, two memory busses, and two I/O busses. These machines have been connected intermittently into the ARPA Network for testing purposes and operational installation in the network is anticipated shortly. A single processor has been running on the network for an extended period in order to validate performance during routine operation. Three Satellite IMP configurations[9] are presently under construction as well as a non-IMP configuration designed to provide highly reliable preprocessing and forwarding of seismic data to processing and storage centers.

RELIABILITY GOALS

Since the term "reliable system" can have many different meanings, it is important to establish clearly just what we are and what we are not trying to achieve. We are not trying to build a non-failing device (as in Reference 10); instead, we are trying to build a system which will recuperate automatically within seconds, or at most minutes, following a failure. Furthermore, we want the system to survive not only transient failures but also solid failures of any single component. In many cases (such as the IMP job) it is not necessary to operate continuously and perfectly; it suffices to operate correctly most of the time so long as outages are infrequent, kept brief, and fixed without human intervention.

How one copes with infrequent brief outages depends on what one is trying to do. For tasks not tightly coupled to real-time requirements (e.g., for many large numerical computations), a simple device is to choose checkpoints at which to record the state of the system so that one can always recover by restarting from the checkpoint just preceding an outage.[11,12] The IMP system happens to be embedded in a larger system which is quite forgiving. (This is not an uncommon situation.) Thus brief outages of a few seconds are tolerated easily, and outages of many seconds, while causing the particular node to become temporarily unusable, will not in general jeopardize operation of the network as a whole.

Occasionally, despite all efforts, the system will break so catastrophically that it will be unable to recover. Our goal is to reduce the probability of such total system failure to the probability of a multiple hardware failure. We do not try to protect against all possible errors; some of our procedures will fail to detect errors whose probability of occurrence is sufficiently low. For example, all code is periodically checksummed using a 16-bit checksum. A failure that does not disturb the validity of the checksum may not be detected. We do not mind if a failure renders large sections of the machine unusable or inaccessible, providing enough remains to run the system. The presence of runnable hardware, however, is not sufficient to guarantee that operation will be resumed; in addition, the software must be able to survive the transients accompanying the failure and adapt to the remaining hardware. This may include combating and overcoming active failures (for example, when an element such as a processor goes berserk and repeatedly writes meaningless data into memory).

All code is presumed to be debugged—i.e., all frequently occurring problems will have been fixed. On the other hand, we must be able to survive infrequent bugs even when they randomly destroy code, data structures, etc.

In order to avoid complete system failure, a failed component must be repaired or replaced before its backup also breaks. The system must therefore report all failures. The actual repair and/or replacement will of course be performed by humans, but this will generally take place long after the system has noted the failure and reconfigures itself to bypass the failed module. The ratio of mean-time-to-repair to mean-time-between-failures will determine overall system reliability. It must also be possible to remove and replace any component while the system continues to run. Finally, the system should absorb repaired or newly introduced parts gracefully.

STRATEGIES

In order to understand our system it is convenient to consider the strategies used to achieve our goals in two parts which more or less parallel the traditional division into hardware and software. The first part provides hardware that will survive any single failure, even a solid one, in such a way as to leave a potentially runnable machine intact (potentially in that it may need resetting, reloading, etc.). The second part provides all of the facilities necessary to survive any and all transients stemming from the failure and to adapt to running in the new hardware configuration.

Appropriate hardware

We have two basic strategies in providing the hardware. The first is to include extra copies of every vital hardware resource. The second is to provide sufficient isolation between the copies so that any single component failure will impair only one copy.

To increase effective bandwidth in multiprocessors, multiple copies of heavily utilized resources are normally provided. For reliability, however, *all* resources critical to running the algorithm are duplicated. Where possible the system utilizes these extra resources to increase the bandwidth of the system.

It is not sufficient merely to provide duplicate copies of a particular resource; we must also be sure that the copies are not dependent on any common resource. Thus, for example, in addition to providing multiple memories, we also include logically independent, physically modular, multiple busses on which the memories are distributed. Each bus has its own power supply and cooling, and may be disconnected and removed from the racks for servicing while the rest of the machine continues to run.

All central system resources, such as the real time clock and the PID, are duplicated on at least two separate I/O busses. All connections between bus pairs are provided by separate bus couplers so that a coupler failure can disable at most the two busses it is connecting.

Non-central resources, such as individual I/O interfaces, are generally less critical. Provision has been made, however, to connect important lines to two identical interface units (on separate I/O busses) either of which may be selected for use by the program.

To adapt to different hardware configurations, the software must be able to determine what hardware resources are available to it. We have made it convenient to search for and locate those resources which are present and determine the type and parameters of those which are found.

To allow for active failures, all bus couplers have a program-controllable switch that inhibits transactions via that coupler. Thus, a bus may be effectively "amputated" by turning off all couplers from that bus. This mechanism is protected from capricious use by requiring a particular data word (a password) to be stored in a control register of the bus coupler. Naturally an amputated processor is prevented from accessing these passwords.

Finally, although a common reset line is normally considered essential, we have avoided such a line since a single failure on its driver could jeopardize the entire system. There is thus no central point (not even a single power switch) where one can gain control of the entire system at once. Instead, we rely on resetting a section at a time using passwords.

Software survival

With the above features, the Pluribus hardware can experience any single component failure and still present a runnable system. One must assume that as a consequence of a failure, the program may have been destroyed, the processors halted, and the hardware put in some hung state needing to be reset. We now investigate the means used to restore the algorithm to operation after a failure. The various techniques for doing this may be classified under three broad strategies: keep it simple, worry about redundancy, and use watchdog timers throughout.

Simplicity

It is always good to keep a system simple, for then one has a fighting chance to make it work. We describe here three system constraints imposed in the name of simplicity.

First, as mentioned above, we insist that all processors be identical and equal: they are viewed only as resources used to advance the algorithm. Each should be able to do any system task; none should be singled out (except momentarily) for a particular function. The important thing is the algorithm. With this view it is clear that it is simplest if the algorithm is accessible to all processors of the system. A consequence of this is that the full power of the machine can be brought to bear on the part of the algorithm which is busiest at a given time.

One might argue that for some systems it is in fact simpler (or more efficient) to specialize processors to specific tasks. One could, in such a case, then duplicate each different type for reliability. With that approach, however, one must worry about the recovery of several different types of units, and all the possible interactions between them. We consider the recovery problem for a group of identical machines formidable enough.

One consequence of treating all processors equally is that a program can be debugged on a single machine up to the point where the multiple machine interaction matters. Once this has been done, we have found that processor interaction does not present a severe additional debugging problem. On the other hand, finding routine software bugs when a dozen machines are running is a difficult problem.

A second characteristic of our system which arose from a desire to keep things simple is passivity. We use the terms active and passive to describe communication between subsystems in which the receiver is expected to put aside what it is doing and respond. The quicker the required response, the more active the interaction. In general, the more passive the communication, the simpler the receiver can be, because it can wait until a convenient time to process the communication. On the other hand the slower response may complicate things for the sender. We believe that there is a net gain in using more passive systems. An example of this is our decision to make the task disbursing mechanism (the PID) a passive device. Neither the hardware interfaces nor other processors tell a processor what to do; rather, processors ask the PID what should be done next. There are some costs to such a passive system. The resulting slower responsiveness has necessitated additional buffering in some of our interfaces. In addition, the program must regularly break from tasks being executed to check the PID for more important tasks.

The alternatives, however, are far worse. In a more active system, for example one which uses classical priority interrupts, it is difficult to decide which processor to switch to the new task. Furthermore, it is almost impossible to preserve the context of a processor[13] while making such a switch because of the interaction with the resource interlocking system. The possibilities for deadlocks are frightening, and the general mechanism to resolve them cumbersome. With a passive system a processor finishes one task before requesting the next, thus guaranteeing that task switching occurs at a time when there is little context, e.g., no resources are locked.

Passive systems are more reliable for another reason: namely, the recovery mechanisms tend to be far simpler than those for active systems.

As a third example of simplicity we introduce the notion of a reliability subsystem. A reliability subsystem is a part of the overall system which is verified as a unit. A subsystem may include a related set of hardware, program, and/or data structures. The boundaries of these reliability subsystems are not necessarily related at all to the boundaries of the hardware subsystems (processors, busses, memories, etc.) described earlier. The entire system is broken into these subsystems, which verify one another in an orderly fashion.

The subsystems are cleanly bounded with well-defined interfaces. They are self-contained in that each includes a self-test mechanism and reset capability. They are isolated in that all communication between subsystems takes place passively via data structures. Complete interlocking is provided at the boundary of every subsystem so that the subsystems can operate asynchronously with respect to one another.

The monitoring of one subsystem by another is performed using timer modules, as discussed below. These timer modules guarantee that the self-test mechanism of each subsystem operates, and this in turn guarantees that the entire subsystem is operating properly.

Redundancy

Redundancy is simultaneously a blessing and a curse. It occurs in the hardware and the software, and in both control and data paths. We deliberately introduce redundancy to provide reliability and to promote efficiency, and it frequently occurs because it is a natural way to build things. On the other hand the mere existence of redundancy implies a possible disagreement between the versions of the information. Such inconsistencies usually lead to erroneous behavior, and often persist for long periods.

It was not until we adopted a strategy of systematically searching out and identifying all the redundancy in every subsystem that we succeeded in making the subsystems reliable. This process therefore constitutes one of our three basic strategies for constructing robust software.

We use the term redundancy here in a somewhat subtle sense, not only for cases in which the same information is stored in two places, but also when two stored pieces of information each imply a common fact although neither is necessarily sufficient to imply the other.

There are several methods of dealing with redundancy. The first and best is to eliminate it, and always refer to a single copy of the information. When we choose not to eliminate it, we can check the redundancy and explicitly detect and correct any inconsistencies. It does not really matter how this is done as the system is recovering from a failure anyway. What is important is to resolve the inconsistency and keep the algorithm moving. Sometimes it is too difficult to test for inconsistency; then timers can be used as discussed in the next section.

Let us consider a few examples of redundancy to make these ideas more concrete.

- A buffer holding a message to be processed, and a pointer to the buffer on a "to be processed" queue—if the buffer and queue are inconsistent, the buffer will not be processed. Each buffer has its own timer and if not processed in a reasonable time, it will be replaced on the queue.
- A device requesting a bus cycle, and a request-capturing flip-flop in the bus arbiter—if the arbiter and device disagree, the bus may hang. A timer resets the bus after one second of inactivity.
- One processor seeing a memory word at a particular system address and another seeing the same word at

the same address—the software watches for inconsistencies and when they occur declares the memory or one of the processors unusable.

- The PID level used by a particular device and the device serviced in response to that level—the PID level(s) used by each device are program-readable. A process periodically reads them and forces the tables driving the program's response to agree.

TABLE I—Major Subsystems and their Functions

IMP SYSTEM: Watches network behavior - will not cooperate with irresponsible network behavior.

IMP SYSTEM RELIABILITY: Watches IMP SYSTEM (data structures mostly).

CONSENSUS: Watches IMP SYSTEM RELIABILITY, verifies all Common Memory Code (via checksum), watches each processor, finds all usable hardware resources (interfaces, PIDs, memory, processors, etc.), tests each and creates a table of good ones. Makes spare copies of code.

INDIVIDUAL: Watches CONSENSUS, finds all memory and processors it considers usable, determines where the Consensus is communicating and tries to join it.

CODE TESTER: Watches INDIVIDUAL, verifies all Local Memory Code (via a checksum), guarantees control and lock mechanisms.

BUS TIMER + 60Hz INTERRUPT: Watches CODE TESTER, guarantees bus activity.

Timers

We have adopted a uniform structure for implementing a monitoring function between reliability subsystems based on watchdog timers. Consider a subsystem which is being monitored. We design such a subsystem to cycle with a characteristic time constant and insist that a complete self-consistency check be included within every cycle. Regular passage through this cycle therefore is sufficient indication of correct operation of the subsystem. If excessive time goes by without passage through the cycle, it implies that the subsystem has had a failure from which it has not been able to recover by itself. The mechanism for monitoring the cycle is a timer which is restarted by every passage through the cycle. We have both hardware and software timers ranging from five microseconds to two minutes in duration. Another subsystem can monitor this timer and take corrective action if it ever runs out. To avoid the necessity for subsystems to be aware of one another's internal structure, each subsystem includes a reset mechanism which may be externally activated. Thus corrective action consists merely of invoking this reset. The reset algorithm is assumed to work although a particular incarnation in code may fail because it gets damaged. In such a case another subsystem (the code checksummer) will shortly repair the damage.

Note that we have introduced an active element into our otherwise totally passive system. These resets constitute the only active elements and furthermore are invoked only after a failure has occurred. This approach seems to provide for the maximum isolation between subsystems.

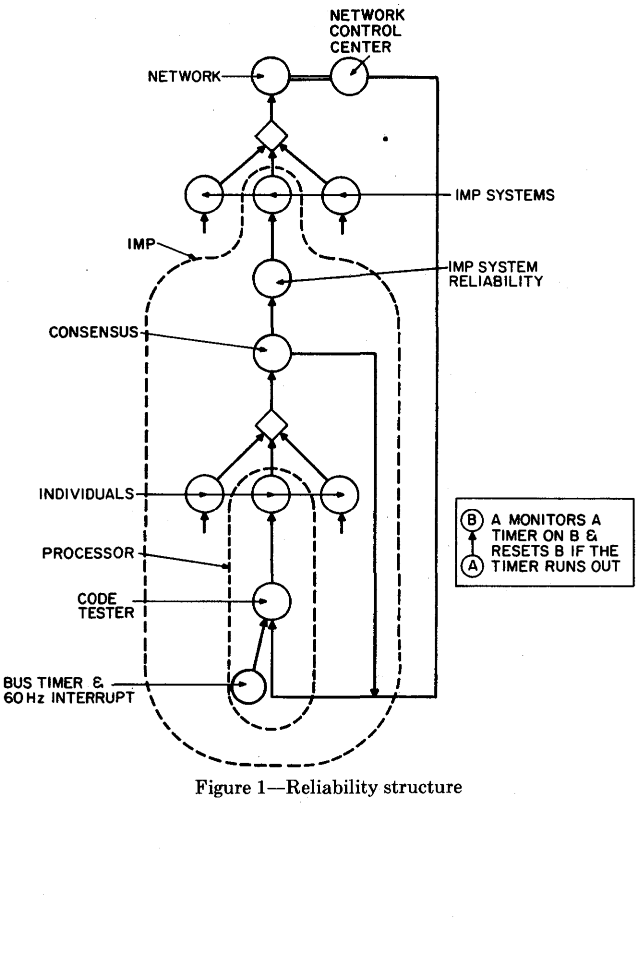


Figure 1—Reliability structure

SYSTEM RELIABILITY STRUCTURE

In the previous section we described a mechanism whereby one subsystem can monitor another. Our system consists of a chain of subsystems in which each subsystem monitors the next member of the chain. Figure 1 and Table I show this structure in the system we have built for the IMP. An efficient way to build such a chain is to have lower subsystems provide and guarantee some important environmental feature used by higher level systems. For example, a low level in our chain guarantees the integrity of code for higher levels which thus assume the correctness of code. Such a system is vulnerable only at its bottom. (We are assuming here that we have runnable hardware although it may be in a bad state, requiring resetting.) The code tester level (see Figure 1) serves three functions: first, it checksums all low level code (including itself); second, it insures that control is operating properly, i.e., that all subsystems are receiving a share of the processors' attention; third, it guarantees that locks do not hang up. It thus guarantees the most basic features for all higher levels. These will, in turn, provide further environmental features, such as a list of working memory areas, I/O devices, etc., to still higher levels. The method by which the code tester subsystem itself is monitored and reset will be discussed shortly.

The mechanisms we have described ensure that the separate processor subsystems have a satisfactory local environment in which to work. Before they can work together to run the main system it is necessary that a common environment be established for all processors. We call the process of reaching an agreement about this environment "forming a consensus", and we dub the group of agreeing processors the Consensus. The work done by the Consensus is in fact performed by individual processors communicating via common memory, but the coordination and discipline imposed on Consensus members make them behave like a single logical entity. An example of a task requiring consensus is the identification of usable common memory and the assignment of functions (code, variables, buffers, etc.) to particular pages. The members of the Consensus will not in general agree in their view of the environment, as for example when a broken bus coupler blinds one member to a segment of common memory. In this case the Consensus, including the processor with the broken coupler, will agree to run the main system without that processor.

The Consensus maintains a timer for every processor in the system, whether or not the processor is working. The Consensus will count down these timers in order to eliminate uncooperative or dead processors. In order to join the Consensus, a processor need merely register its desire to join by holding off its timer. Within the individual processors it is the code tester subsystem which holds off the timer.

The Consensus, then, acting as a group, provides the monitoring mechanism for the individuals as shown by the feedback monitoring path in Figure 1. This monitoring mechanism run by the Consensus includes the usual reset capability which in this case means reloading the individual's local memory and restarting the processor. Since all of the processors have identical memories, reloading is not difficult. We provide (password protected) paths whereby any processor can reset, reload, and restart any other processor. This reliance on the Consensus is indeed vulnerable to a simultaneous transient failure of all processors. However, the Network Control Center has access to these same reset and reload facilities and these enable it to perform the reload function remotely (a path also shown in the figure).

Thus the Consensus and/or Network Control Center are the ultimate guarantors of the lowest level subsystem. While this process is sufficient it is sometimes slow. For many cases in which the Consensus is disabled (as for example when all of the processors halt), a simpler reset without reloading will suffice. For this reason we have provided a simpler and more immediate (if redundant) mechanism in each processor for resetting the control and lock systems. We implement this mechanism in software with the assistance of a 60Hz interrupt and a one-second timer on the bus. Together these provide a somewhat vulnerable but much quicker alternative to the more ponderous NCC/Consensus resets.

There is a problem about what area of common memory the processors should use in which to form the Consensus, since failures may make any predetermined system address inaccessible. To allow for this, sufficient communication is maintained in all pages of common memory to reach agreement both as to which processors are in the Consensus and where further communication is to take place.

To protect itself from broken processors, the Consensus amputates all processors which do not succeed in joining it. There is a conflict between this need to protect itself and the need to admit new or healed processors into the Consensus. The amputation barrier is therefore lowered for a brief period each time the Consensus tries to restart a processor. This restart is in fact the reset based on the timer held off by the code tester subsystem, as discussed above. In the case of certain active failures, even this brief relaxation may cause trouble. In these cases the Consensus will decide to keep the barrier up continuously.

Certain active failures may prevent the formation of a consensus. In such a situation each processor will behave as if it were a Consensus (of one) and will try to amputate all other processors. At the point when the actively failing component is amputated, the remaining processors will be able to form a consensus. From this point the system behaves as described above.

Further up in the figure there is another joining of independent units, namely IMPs joining to form the network. The analogy here is incomplete because the ARPA Network was not built with these concepts in mind. There is collective behavior, e.g., routing, and individual behavior which accepts collective decisions only after they pass reasonability tests. However, the reliability features of the network are concentrated in the Network Control Center, which depends on the continual presence of human operators for successful operation. It is correspondingly powerful, resourceful, and erratic in its behavior.

SOME EXAMPLES OF FAILURES

In order to explain in more practical terms some of the reliability mechanisms, we now discuss a number of specific failures and describe the methods which detect and repair the resulting damage. In each case, we focus on the component that failed and the particular mechanism that takes care of that failure. Derivative failures may well take place, and other mechanisms will handle these, since all mechanisms operate all the time.

These examples are set in the context of the IMP application and the severity of their direct consequences rated on the following scale:

1. Momentary slowdown—no data loss
2. Loss of data (a network message)
3. Temporary loss of some IMP function (a network link)
4. Momentary total IMP outage with local self-recovery
5. Outage requiring reloading via the network
6. Failure requiring human insight for debugging.

Example 1. Suppose first that a bus coupler experiences a transient failure on a single reference to common memory, which leaves one word of common memory with the wrong contents but correct parity. Suppose further that the failure is subtle, in the sense that there is no obvious ill effect on processor control, like halting or looping, which will be caught by lower level mechanisms. We will focus first on examples which cause minimal disruption and where detection and gentle recovery are the primary concerns. We consider four examples of transient memory failures:

Example 1.a Suppose that a word of text in one of the messages we are delivering becomes smashed. There is a checksum on all messages and the network will notice at one of its checkpoints that the message has gone bad. The source will be prompted to send a new copy. (Severity 2)

Example 1.b Near the heart of our system is a queue of unused buffers called the free list. Suppose the failure is in the structure of this queue. The system explicitly tests for both a looped queue and wrong things on the queue. A more subtle form of error occurs when some buffers which should be on the queue are missing from it. Our system is designed so that a buffer should be removed from the free list for at most two minutes at a time. A timer is maintained on each buffer, which is restarted whenever the buffer returns to the free list. Should any timer ever run out, its buffer is forced back onto the free list. The result of this failure will be a degradation of system performance as it attempts to run with fewer buffers for a short while, followed by complete recovery within two minutes. The IMP will stay up and no messages will be lost. (Severity 1)

Example 1.c Suppose that one of the locks on a resource is broken so that it wrongly locks the resource. Any subsystem which tries to use the resource will put a processor into a tight loop waiting for the resource to become free. In about 1/15 sec. this will cause the processor's timer, driven off its 60Hz clock interrupt, to run out. Upon investigation, the program will notice that the subsystem is waiting for a locked resource, and arbitrarily unlocks it. Aside from the 1/15 sec. pause, the system will be unaffected by the transient. (Compare the simplicity of this scheme with Reference 14.) (Severity 1)

Example 1.d Suppose now that a failure strikes common memory holding code, and that the trouble is subtle—either the code is not run often or the change has no immediate drastic effect. In a few seconds the processors will begin to notice that the checksum on that piece of code is bad and stop running it. Shortly the whole Consensus will agree, and will switch over to use the memory holding the spare copy of that code. Unless the broken code has already caused some other trouble, the problem is thereby fixed, with only momentary slowdown. (Severity 1)

Example 2. Suppose a processor fails by suddenly and permanently stopping. The immediate effect will be that some task will be left half done, with a high probability that some resource is locked. This looks to the system like a data failure, as in examples 1.a, 1.b, and 1.c above. The recovery will be identical. In a few seconds the Consensus will notice that the processor has dropped out and processor recovery logic will be invoked. Since the processor is solidly broken the recovery will be unsuccessful, and the system will settle into a mode where every so often recovery is retried. Eventually a repairman will fix the processor, at which time recovery will proceed and the processor will rejoin Consensus. It is hard to predict whether the IMP system will momentarily go down because of the failure; experience indicates that it usually stays up, but our experience is limited to lightly loaded machines. (Severity 2-4)

Example 3. Suppose a power supply for a processor bus breaks. This is similar to the failing processor described above except that both processors on the bus are affected and the processors are given a hardware warning sufficiently far in advance that they can halt cleanly. The system will surely stay up through this failure. (Severity 1)

Example 4. Now consider a case in which some page of common memory ceases to answer when referenced. Each processor will get a hardware trap when it tries to use that memory, forcing it directly to the code which routinely verifies the environment. As a result, the failing memory will be deleted from the memory list by the Consensus and another module will be pressed into service to take its place.

If the failed page contained code, a spare copy will normally be available and a new spare copy will be made if possible. If it contained data, an unused page will be pressed into service. In either case, the system will be reinitialized, momentarily bringing the IMP system down. If the failed page contained the Consensus communication area, a new Consensus must be formed and thus recovery will take a little longer. (Severity 4)

Example 5. Let us now consider a failure of the PID. Suppose that the PID reports a task not previously set. When invoked, each strip checks to make sure that it is reasonable for the strip to be run. If not, another task is sought. Suppose instead that the PID "drops" a task. A special process periodically sets all PID flags independent of what needs to be done. This causes no harm, because superfluous tasks will be ignored (as described above), and serves to pick up such dropped tasks. Thus we have both a consistency check on redundant information and a timer built into our use of the PID. If a PID fails solidly, another PID will be switched in to operate the system. Transient failures cause slowdown; switchover may momentarily bring down the IMP system. (Severity 1, 4)

All of this leads to a slightly different image of the PID. Instead of being the central task disburser, with all processors relying on it to tell them what to do, the PID is a guide, suggesting to processors that if they look in a certain place, they will probably find some useful work to do. The system would in fact run without a PID, albeit much more slowly and inefficiently.

Example 6. Suppose a halt instruction somehow gets planted in common memory and that all processors execute it and stop. There is thus no Consensus left to come to the rescue. Furthermore, 60Hz interrupts are ineffective in a halted processor. After one second of inactivity, the bus arbiter timer will reset the processors, making them once more eligible for 60Hz interrupts which will restart them. Before the broken code is run, it will be checksummed, the discrepancy found, and a spare copy used. (Severity 2-4)

Example 7. Let us consider now what happens when, in common memory, an end test for a storing loop is destroyed, causing each processor to wipe out its 60Hz interrupt code in local memory. In this case not only are there no processors left to help, but the 60Hz interrupt will not help either, since the interrupt code itself is broken. This is a case in which the machine is incapable of rescuing itself and will go off the network as a working node. When the Network Control Center notices that the IMP is no longer up, it will commence an external reload, restoring the IMP to operation. (Severity 5)

Example 8. Consider the case of a processor whose hardware is solidly broken such that it repeatedly stores a zero into a location in common memory. Mechanisms described above will repeatedly fix the changed location, but it is desirable to eliminate the continuing presence of this disrupting influence. The Consensus will notice that one of its number has dropped out and will endeavor to help the errant processor. After a few tries, a longer timer will run out, and the Consensus will take a more drastic action: final amputation. In this case there will be a rather lengthy IMP outage but the system will recover without external help. (Severity 4)

Example 9. One failure from which there is no recovery, either automatic or remote, is a program which impersonates normal behavior but is still somehow incorrect. That is, it holds off the right timers, has a valid checksum, and simulates enough normal behavior so that higher levels (e.g., the NCC) are satisfied. For example, if it were not for the fact that the NCC explicitly checks the version number of the program running in each IMP, a previous, compatible, but obsolete version of the program would exhibit this behavior. (Severity 6)

Example 10. Another class of failures which is hard to isolate and deal with is low-frequency intermittents. Consider the case of a single processor which is broken such that its indexed shift instruction performs incorrectly. Since this instruction only occurs in some infrequently executed procedures, the failure only manifests itself, on the average, once every period t. If t is large, for instance one year, then we can safely disregard the error, since its frequency is in the range of other failures over which we have no control. If it is small, say 100 milliseconds, then the Consensus will isolate the bad processor and excise it. At some intermediate frequency, however, the Consensus will fail to correlate successive failures and will instead treat each as a separate transient. The system will repeatedly fail and recover until some human intervenes. (Severity 6)

RESULTS AND CONCLUSIONS

Some strategies and techniques for building a reliable multiprocessor have been described above. We have, in fact, actually built and programmed such a machine using these strategies. We have found this machine straightforward to debug, both in hardware and software. Furthermore, the system continues to operate when individual power supplies are turned off, when memory locations are altered, when cables and cards are torn out, and through a variety of other failures. We have yet to establish field performance (which must be measured both in rate of recoverable incidents and in rate of unrecoverable failures), but we expect to start gathering this information shortly.

We believe there are many important problems in the world today which could benefit from the principles described here. While we have discussed these principles in terms of a specific application (the IMP), most of the concepts are application independent. We have been able to separate the application code from the reliability subsystems intact in another application of the Pluribus machine.

ACKNOWLEDGMENTS

Many people in addition to the authors have contributed to the ideas described herein, notably Benjamin Barker, John Robinson, David Walden, John McQuillan, and William Mann. In addition, there is a long list of those who helped to bring these machines into existence. Foremost among these are Martin Thrope, David Katsuki and Steven Jeske. The work reported here would not have been possible without the continued support of the ARPA/IPT office. Finally, a word of thanks to Robert Brooks and Julie Moore, who helped to prepare the manuscript.

REFERENCES

1. Riley, W. B., "Minicomputer Networks—A Challenge to Maxicomputers?" *Electronics*, March 29, 1971, pp. 56-62.
2. Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther and D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," *AFIPS Conference Proceedings*, Vol. 36, June 1970, pp. 551-567; also in *Advances in Computer Communications*, W. W. Chu (ed.), Artech House Inc., 1974, pp. 300-316.
3. Roberts, L. G. and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *AFIPS Conference Proceedings*, Vol. 36, June 1970, pp. 543-549.
4. Heart, F. E., S. M. Ornstein, W. R. Crowther and W. B. Barker, "A New Minicomputer/Multiprocessor for the ARPA Network," *AFIPS Conference Proceedings*, Vol. 42, June 1973, pp. 529-537; also in Selected Papers: International Advanced Study Institute, *Computer Communication Networks*, R. L. Grimsdale and F. F. Kuo (eds.) University of Sussex, Brighton, England, September 1973; also in

Advances in Computer Communications, W. W. Chu (ed.), Artech House Inc., 1974, pp. 329-337.

5. Ornstein, S. M., W. B. Barker, R. D. Bressler, W. R. Crowther, F. E. Heart, M. F. Kraley, A. Michel and M. J. Thrope, "The BBN Multiprocessor," *Proceedings of the Seventh Annual Hawaii International Conference on System Sciences*, Honolulu, Hawaii, January 1974, Computer Nets Supplement, pp. 92-95.
6. Crowther, W. R., J. M. McQuillan and D. C. Walden, "Reliability Issues in the ARPA Network," *Proceedings of the ACM/IEEE Third Data Communications Symposium*, November 1973, pp. 159-160.
7. McKenzie, A. A., B. P. Cosell, J. M. McQuillan and M. J. Thrope, "The Network Control Center for the ARPA Network," *Proceedings of the First International Conference on Computer Communication*, Washington, D.C., October 1972, pp. 185-191.
8. Dijkstra, E. W., "Cooperating Sequential Processes," in *Programming Languages*, ed. F. Genuys, Academic Press, London and New York 1968, pp. 43-112.
9. Butterfield, S. C., R. D. Rettberg and D. C. Walden, "The Satellite IMP for the ARPA Network," *Proceedings of the Seventh Annual Hawaii International Conference on System Sciences*, Honolulu, Hawaii, January 1974, Computer Nets Supplement, pp. 70-73.
10. Hopkins, A. L., Jr., "A Fault-Tolerant Information Processing Concept for Space Vehicles," *IEEE Transactions on Computers,* Volume C-20, Number 11, November 1971, pp. 1394-1403.
11. Avizienes, A., G. C. Gilley, F. P. Mathur, D. A. Rennels, I. A. Rohr and D. K. Rubin, "The STAR (Self-Testing and Repairing) Computer: An Investigation of the Theory and Practice of Fault-Tolerant Computer Design," *IEEE Transactions on Computers*, Volume C-20, Number 11, November 1971, pp. 1312-1321.
12. IBM Corporation, *OS Advanced Checkpoint/Restart*, IBM Manual GC28-6708.
13. Gountanis, R. J. and N. L. Viss, "A Method of Processor Selection for Interrupt Handling in a Multiprocessor System," *Proceedings of the IEEE*, Vol. 54, No. 12, December 1966, pp. 1812-1819.
14. Lamport, L., "A New Solution of Dijkstra's Concurrent Programming Problem," *Communications of the ACM*, Volume 17, Number 8, August 1974, pp. 453-455.

Design considerations for a heterogeneous tightly-coupled multiprocessor system

by KENICHIRO NOGUCHI, ISAO OHNISHI and HIROSHI MORITA
Hitachi, Ltd.
Yokohama, Japan

INTRODUCTION

In a multiprocessor system, processors share main memory and a single copy of the operating system in shared main memory controls the entire system. Basically each processor can execute, any of the programs in the system. (This type of multiprocessor system is sometimes called a tightly-coupled multiprocessor system[1,2] to distinguish from another type of multiprocessor system in which each processor has its own main memory and operating system. In this paper a "multiprocessor system" means a "tightly-coupled multiprocessor system" unless otherwise noted.) A multiprocessor system usually consists of identical processors, which have same computing speeds as well as the same functional characteristics. In this paper a more general type of multiprocessor system which consists of processors of different computing speeds are discussed. The component processors are equivalent in the hardware functions but have different performance characteristics. This type of multiprocessor system, a heterogeneous multiprocessor system, has the following merits as compared with a homogeneous multiprocessor.

System organization flexibility

Installations can select appropriate computing power among many combinations of processors which a heterogeneous multiprocessor system provides. For example, if the operating system and the hardware provide the support of a heterogeneous multiprocessor system with two processors of different kinds, it can be expected that it is also possible to organize homogeneous multiprocessor systems with the same kind of processors as well as and single-processor systems. Thus there are five possible processor combinations, provided that there are two kinds of processors. In general, if there are K kinds of processors and the operating system and the hardware provide the support of a heterogeneous multiprocessor system which can contain up to M kinds of processors ($M \leqq K$) and up to N processors in total, then there are

$$NK + \sum_{n=2}^{N} \sum_{m=2}^{\min(n,M)} \binom{K}{m} \cdot \binom{n-1}{m-1}$$

possible processor combinations. If $M=K=2$, it gives $N(N+3)/2$.

System performance improvement

Usually different kinds of processors differ not only in total computing speeds but also in their relative speeds to various computation loads. Some kind of processor may favor the scientific computation loads with floating-point arithmetics rather than the system control functions with logical operations, and *vice versa* for some other kind of processor. It is possible to assign favorite types of computation loads to each kind of processor so that the total processing throughput is improved. This kind of load sharing can also be done in a multiprocessor system without shared main memory, where the load sharing is performed through job scheduling.[3] But more versatile load sharing is possible in a heterogeneous multiprocessor system with shared main memory, because basically each processor can execute any programs in the system. If a processor has no other favorite load because the execution of the favorite load has completed or is suspended by input/output operation, the processor executes any other existing loads in the system.

In this paper, design considerations of the heterogeneous multiprocessor system which were developed for Hitachi's operating system (OS7) are presented. OS7 is a large-scale, general-purpose operating system[4] and can contain up to four processors of HITAC 8700('s) and/or HITAC 8800('s).[5] (So there are 14 possible processor combinations.) Other significant features of OS7 include virtual memory with 2048 mega byte address space for each user and integrated support of batch processing, remote-batch processing, time-sharing processing, and real-time processing under the single operating system.

In the following sections of this paper hardware considerations, software considerations, and performance evaluation of the heterogeneous multiprocessor system are discussed.

HARDWARE CONSIDERATIONS

The hardware features for multiprocessing of H8700 and H8800 are briefly discussed in this section. H8700 and H8800 have the same hardware architecture. They have dynamic address translation (DAT) mechanism, buffer memory (cache), and ring protection mechanism. They have multiprocessing features of inter-processor communication, prefixing, processor synchronization, and configuration control.

For inter-processor communication purposes, the direct control mechanism is provided. It includes the facilities for signaling other processors which are used by programs and the facilities for informing other processors of cancel signals for buffer memory and hardware error information. Each processor of the multiprocessor has unique memory area called prefix area (8 kilo bytes for each processor), which is pointed to by the prefix register. For processor synchronization purposes, a Test and Set (TS) instruction is provided. H8700 and H8800 provide the configuration control function which is controllable by programs for availability and organization flexibility purpose. Each component of the system, i.e., a processor, a memory unit and an Input/Output Processor (IOP), has a configuration register which controls the connection status of the component to other components. The connection and disconnection of the component to and from the system is accomplished by programs through changing the contents of the configuration registers.

To organize a heterogeneous multiprocessor system H8700 and H8800 are designed with the following principles:

- Both processors have identical architecture and identical interfaces to other components.
- All components of the system—processors, memory units, and IOP's—operate asynchronously.

SOFTWARE CONSIDERATIONS

General features for multiprocessing

Before discussing the features of the operating system for the heterogeneous multiprocessor system, the general features for multiprocessing are discussed.

Actions

In OS7 it is intended that supervisory programs are run in enabled status (i.e., allowing interruptions) and in parallel as much as possible. By this approach system responsiveness to external signals and system efficiency can be increased. Besides a task, an *action* is introduced as a new type of process to realize the above objective. An action has its flow of control, has its own status save area in the Action Control Block (ACB), and is usually interruptible. The difference between actions and tasks is that actions have higher dispatching priority than tasks (actions are dispatched by Action Scanner; if Action Scanner finds no action to be processed, control goes to Task Dispatcher), require less overhead to create and destroy, can run in any address space, and execute only supervisory programs. External interruptions such as input/output termination interruptions are processed by actions, including input/output error recovery functions; memory scheduling functions and many other supervisory functions are performed by actions.

Locks

In the multiprocessor environment, the locks are necessary to serialize the execution of critical portions in the disabled (i.e., not allowing interruptions) supervisory programs. The simplest method is to introduce a single lock which serializes the execution of all disabled supervisor programs, but all this creates a performance bottleneck when heavy supervisory program services are required. In OS7, in order to minimize the loss time due to locks, locks are designed to correspond to each of the supervisor resources, e.g., some types of system control blocks or critical programs such as Task Dispatcher. There are more than 50 multiprocessor locks.

In OS7 locks are classified into two types, a spin type and a control relinquish type. If a processor has failed to get a spin type lock by a Test and Set instruction, it repeats the instruction until it will succeed. If a processor has failed to get a control relinquish type lock, the current process (task or action) releases control and process switch occurs. The control release type locks can be used in the enabled programs and in the programs where mapping faults may occur.

Inter-processor communications

In the multiprocessor environment, there are many cases in which inter-processor communication is necessary. Since main memory is shared by the processors, the means of signaling other processors is sufficient. This is done by issuing a Write Direct instruction to the destination processors, which causes an external interruption in the destination processor. Signaling other processors is performed in the following cases:

- when the master processor (the processor from which initial program loading (IPL) has been initiated) wakes up other processors in the system at system initiation time;
- when Task Dispatcher finds an idle processor at the time some task has attained ready status, or when Task Dispatcher finds a processor which is executing a lower priority task than the task which has attained ready status (in the latter case rescheduling of tasks is requested);
- when Action Scanner finds an action which is to be executed by another processor (there are two types of actions, common actions which any processor can execute and fixed-processor actions which are tied to specific processors);
- when the paging control routine detects that address translation information of the page which it is about to invalidate is contained in another processor.

Features for heterogeneous multiprocessor

In this section the features of the operating system for the heterogeneous multiprocessor system are discussed. They include asymmetric task scheduling, processor timer conversion, and input/output control.

Asymmetric task scheduling

In the heterogeneous multiprocessor system of OS7, there are two models of processors, H8700 and H8800. They differ in computing speeds and performance characteristics. To improve system performance, and to meet turnaround time requirements, Task Dispatcher performs asymmetric task scheduling.

Because H8800 is faster than H8700, Task Dispatcher ensures that H8800 will not become idle when H8700 is executing a task. When H8800 finds no ready task, it examines the status of H8700's. If some of them are executing tasks, H8800 takes over one of them. To accomplish this, direct control signals are issued first from H8800 to H8700 requesting task take over, and then from H8700 back to H8800 reporting that H8700 has released the task.

H8800 and H8700 have different performance characteristics. H8800 is much faster than H8700 in arithmetic operations rather than in logical operations and memory to memory operations. Task Dispatcher reflects these characteristics to improve the system performance, i.e., H8800 and H8700 are controlled to have different task preference. User tasks mainly perform arithmetic operations especially in scientific applications and system tasks (tasks which control system activities, e.g., job scheduling, console input/output control, system input/output control, TSS control, communication control, etc.) mainly perform logical operations, memory to memory operations and input/output operations. Therefore, H8800 is controlled to prefer user tasks to system tasks, and the reverse for H8700. Internally effective priority of a task is adopted which is determined by the task's inherent priority, the type of the task and the model of the processor which is about to execute the task. User tasks have higher effective priorities than system tasks in H8800 and *vice versa* in H8700 (Figure 1).

In the heterogeneous multiprocessor environment, there is a demand that some job be processed with the shortest turnaround time, that is, processed by the fastest processor. In OS7 an express task is introduced. An express task has a highest priority among user tasks and is executed only by H8800.

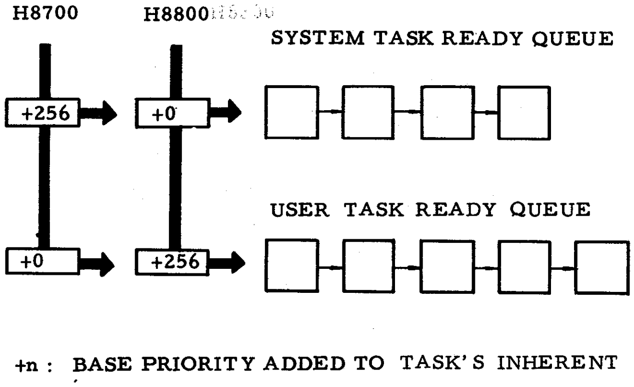


Figure 1—Task preference of HITAC 8700/8800 in asymmetric task scheduling

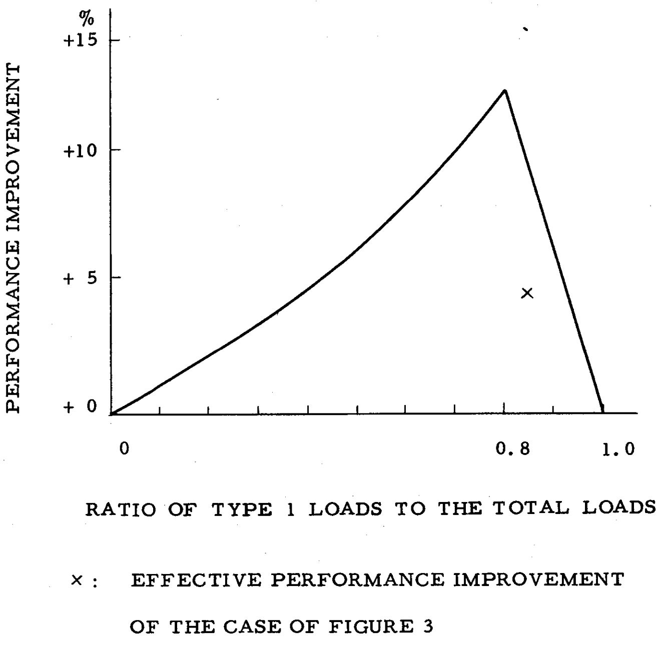


Figure 2—Theoretical performance improvement of asymmetric load scheduling (The case that $\mu_1:\mu_2:\nu_1:\nu_2=1:2:4:4$ in Equation (3))

Processor timer conversion

In the operating system, the processor time is measured for many purposes; for accounting, for job termination by processor time limit and for time slicing control. User programs may also measure it. In the heterogeneous multiprocessor system, to adjust for processor speed differences, processor timer conversion is performed. One processor model is selected as a standard processor. Every processor time in the system is expressed in the standard processor time, and processor time of other models are converted to the standard processor time. When processor model switch occurs on a task, processor timer conversion control is performed. In OS7, H8700 time is converted by a constant factor ($\frac{1}{3}\sim\frac{1}{4}$ is usually used) to H8800 time. Strictly speaking, this value varies depending on the computation loads. In OS7, the installation can change this value.

Input/output control

In OS7, a channel is controlled by one processor. Input/output instructions to the channel are issued from that processor and termination interruptions come to the processor. The correspondence of a channel to its controlling processor is established at the system generation time. When an input/output request to some channel is issued on another processor than the controlling processor of the channel, the request is chained to the channel request queue. If the channel is busy at that moment, no other action is necessary. The request will be picked up after termination interruption occurs. If the channel is not busy, it is necessary to inform the

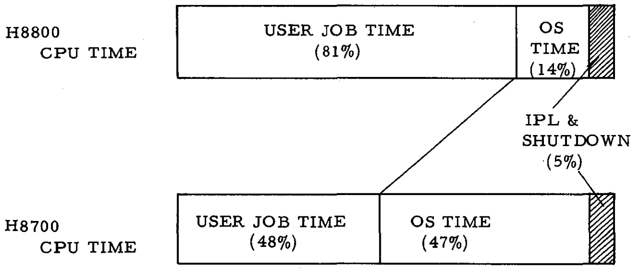


Figure 3—Processor time usage at the Computer Centre, the University of Tokyo (Total of Feb. 1974)[6]

controlling processor that a channel request has arrived. This is performed by creating an *action* which is fixed to the processor and processes channel requests. The creation of the action is made known by a direct control signal, and the informed processor (controlling processor of the channel) executes the action.

In the heterogeneous multiprocessor environment, all channels can be attached to H8700's, thus all input/output operations can be performed by H8700. This is another type of load sharing in the heterogeneous multiprocessor.

PERFORMANCE EVALUATION

Analysis of performance improvement

The performance improvement of a heterogeneous multiprocessor system which is gained by an appropriate load sharing is analyzed. In this analysis it is assumed that there are two types of computation loads, type 1 loads and type 2 loads, and a heterogeneous multiprocessor system consists of two kinds of processors, type 1 processors and type 2 processors. The "effective average instruction execution times" of type 1 processors when executing type 1 loads and type 2 loads are μ_1, and μ_2, respectively; also for type 2 processors, ν_1 and ν_2. The "effective average instruction execution time" of processors is given by dividing the average instruction execution time of one processor by the number of processors. The proportions of type 1 loads and type 2 loads are x and $1-x$, respectively. (The amounts of loads are measured in terms of instructions to be executed.)

- The case that no asymmetric load scheduling is performed. In this case both types of processors execute higher priority loads first, hence the effective instruction execution time of the total system, T_{NA}, is given by

$$T_{NA}=\frac{\mu_1\nu_1}{\mu_1+\nu_1}x+\frac{\mu_2\nu_2}{\mu_2+\nu_2}(1-x) \qquad (1)$$

- The case that asymmetric load scheduling is performed. The following scheduling is assumed; 1) type 1 processors execute type 1 loads first and type 2 processors execute type 2 loads first; 2) when some type of loads are exhausted, both types of processors execute remaining loads. In this case the effective instruction execution time of the total system, T_A, is given, depending on the proportion of loads, by

$$T_A=\begin{cases}\dfrac{\mu_1(\nu_2+(\nu_1-\nu_2)x)}{\mu_1+\nu_1} & \text{for } x\geqq\dfrac{\nu_2}{\mu_1+\nu_2}\\[2ex] \dfrac{\nu_2(\mu_2+(\mu_1-\mu_2)x)}{\mu_2+\nu_2} & \text{for } x<\dfrac{\nu_2}{\mu_1+\nu_2}\end{cases} \qquad (2)$$

- The degree of performance improvement. The performance is improved by the asymmetric load scheduling

(i.e., $T_A<T_{NA}$)

when $$\frac{\mu_2}{\mu_1}>\frac{\nu_2}{\nu_1}$$

The degree of performance improvement, T_{NA}/T_A, is given by

$$\frac{T_{NA}}{T_A}=\begin{cases}1+\dfrac{(\mu_2\nu_1-\mu_1\nu_2)\nu_2(1-x)}{\mu_1(\mu_2+\nu_2)(\nu_2+(\nu_1-\nu_2)x)} & \text{for } x\geqq\dfrac{\nu_2}{\mu_1+\nu_2}\\[2ex] 1+\dfrac{(\mu_2\nu_1-\mu_1\nu_2)\mu_1x}{\nu_2(\mu_1+\nu_1)(\mu_2+(\mu_1-\mu_2)x)} & \text{for } x<\dfrac{\nu_2}{\mu_1+\nu_2}\end{cases} \qquad (3)$$

These results mean that when performance characteristics of two kinds of processors differ (i.e. $\mu_1/\mu_2\neq\nu_1/\nu_2$), the asymmetric load scheduling improves system performance and the degree of improvement is given by (3).

As an example, T_{NA}/T_A is shown in Figure 2 in the case the ratio of μ_1:μ_2:ν_1:ν_2 is 1:2:4:4.

Discussion on measurement data

Figure 3 shows the effect of asymmetric scheduling of the existing system. This was measured at the Computer Centre, the University of Tokyo.[6] The system consists of two H8800's and two H8700's. H8800's mainly execute user tasks. H8700's execute input/output operations and system tasks first, and then execute user tasks. Since some portions of the operating system programs are executed on the user task, all the H8800 CPU time is not user job time. In the measurement data OS time includes 2∼3 percent of idle time.

The ratio of average instruction execution times of H8800 and H8700, for executing user programs in the scientific applications and OS programs is approximately equal to the ratio presented in the previous example. Hence the performance improvement gained by the asymmetric load scheduling is estimated to be 5 percent. In this system the proportion of the operating system execution is small. If the proportion of OS loads increases, larger degree of performance improvement can be gained (see Figure 3).

SUMMARY

The heterogeneous tightly-coupled multiprocessor system is the extension of the homogeneous tightly-coupled multiprocessor system. The heterogeneous multiprocessor system has the merits of system organization flexibility and performance improvement with appropriate load sharing. Features for the heterogeneous multiprocessor system were described and the performance improvement of the asymmetric load scheduling analyzed and evaluated. OS7 provides the support of the heterogeneous multiprocessor system for HITAC 8700('s) and HITAC 8800('s). It has been in operation, as heterogeneous multiprocessor systems, since January 1973.

ACKNOWLEDGMENT

Authors wish to acknowledge Prof. H. Ishida of the University of Tokyo, who always gives valuable suggestions, and many members of Hitachi, Ltd., who have participated in the development of OS7 and HITAC 8700/8800.

REFERENCES

1. Mackinnon, R. A., "Advanced Function Extended with Tightly-Coupled Multiprocessing," *IBM Systems Journal,* Vol. 13, No. 1, pp. 32-59, 1974.
2. Arnold, J. S., D. P., Casey, and R. H., McKinstry, "Design of Tightly-Coupled Multiprocessing Programming," *IBM Systems Journal,* Vol. 13, No. 1, pp. 60-87, 1974.
3. Liu, J. W. S., and C. L., Liu, "Bounds on Scheduling Algorithms for Heterogeneous Computing Systems," *Information Processing* 74, North-Holland Publishing Company, pp. 349-353, 1974.
4. Ohnishi, I., S., Totsune, and H., Ishida, "Command Language in OS7," *Proceedings of the IFIP Working Conference on Command Languages,* July 1974.
5. Nakazawa, K., K., Murata, K., Ishihara, H., Iwakami, H., Horikoshi, H., Nishino, and K., Noda, "The Development of the High Speed National Project Computer System," *Proceedings of the first USA-JAPAN Computer Conference,* pp. 173-181, Oct. 1972.
6. Ishida, H., "A 4-CPU Multiprocessor System of the University of Tokyo," *Journal of the Information Processing Society of Japan,* Vol. 15, No. 7, pp. 534-541, July 1974 (in Japanese).

Microprocessor—Based multiprocessor ring structured network

by HOO-MIN D. TOONG
Massachusetts Institute of Technology
Cambridge, Massachusetts

SUMMARY

A good deal of interest has been generated by the use of multiprocessor distributed architecture networks for resource sharing, for load equalization, and for interprocessor message communications. Such networks generally employ node processing elements that fall into several distinct categories: general-purpose homogeneous processors, general-purpose nonhomogeneous processors, array computers, and pipe-line computers. Topology spans a multitude of configurations from generalized graph structures, to tree and star networks, to multiple-bus and time-shared bus systems.

A particular ring-structured network of microprocessor controlled node elements, intended to realize resource sharing and high speed interprocessor communication among a pool of hardware and software resources, has been implemented. The microprocessor approach allows a core hardware/software system of general-purpose homogeneous processors to be developed independent of individual characteristics of the nonhomogeneous set of resources which it will service. Adaptation of any node to the particular device(s) or resource which it must manage is accomplished primarily through microprocessor program development. Complete node interchangeability and automatic ring expansion or contraction is readily achieved.

Each resource (such as disk, computer, line printer, terminal) is interfaced to a network node which communicates with other nodes via wideband dedicated serial lines. Network interfaces at each node are responsible for all aspects of message transmission and reception, thus freeing the microprocessors to manage system resources. Double frequency data encoding is accomplished at a 7 Mbit/sec rate with phase-locked loop techniques employed at individual nodes for clock regeneration. Data is continually inspected at each node in a nonbuffered fashion for encoding violations as a first-order check on data validity; standard error checking mechanisms are also employed to determine message correctness. Because of phase-gap generation in such a ring, the node that establishes network timing must utilize a simultaneous read/write FIFO buffer to allow smooth flow of information around the entire ring.

Because all nodes are identical in structure and message capabilities from the network viewpoint, ring control for the purposes of message transmission is passed from node to node. Under appropriate conditions such as node failure or data corruption threatening network control, resynchronization may be invoked by any node; during this period of reinitialization control is removed from all nodes and network activity is ceased temporarily. Network timing is subsequently reestablished and the ring picks up where it left off. Appropriate logic within the node interface handles retransmission attempts whenever error conditions cause messages to be received improperly. Hardware assemblies at each node comprise a CPU, ROM, RAM, and resource interfaces. An additional memory controller accomplishes DMA type transfers as well as buffering messages before and after transmission.

Information management is handled via a network file system responsible for directory maintenance, access rights, and appropriate allocation or deallocation of node data bases. The file system is hierarchical allowing an inner program structure to handle all network file communications while an outer structure is tailored to handle node-resource communication.

Trends in database management—1975

by CHARLES W. BACHMAN
Honeywell Information Systems, Inc.
Waltham, Massachusetts

The subject of database management systems is receiving attention at an accelerating rate. The work of the COBOL Data Base Task Group (DBTG) has achieved major acceptance. This acceptance can be measured in terms of the number of computer systems for which DBTG software is now available, and the number of customers who have successfully installed application systems based upon this software. While the specifications are yet imperfect and development committees will be refining them for years, it appears to this author that a major step in the evolution of computerized information systems has been accomplished. Integrated database systems are here to stay.

Given DBTG as a fact, my theme is to suggest where database management systems might go from here. These remarks should not be considered as a survey of the possibilities, but rather as a description of three areas where the author has personal knowledge and which seem to have promise of a strong affect upon the future database systems. The first such area is development of a three schema technology to replace the two schemas of DBTG. This technology is being developed by ANSI/X3/SPARC/Study Group Database Systems. It is being driven by the thrust for data independence. The second area is concerned with the very active debate between the supporters of data-structure-set models which underly the DBTG systems, and the relational model. The third area relates to the development of hardware to assist in the repetitive operations characteristic of database systems. The database features of the new Honeywell Series 60/Level 64 computer is the specific case to be discussed.

The ANSI/X3/SPARC/Study Group Database Systems[1] was activated in November 1972 to examine the question of whether database systems were ready for standardization. To-date, this group has concentrated upon the question, "What is the basic architecture of the database systems necessary to support the user requirements of the late 1970's and the 1980's? The first requirement was to provide a series of alternate data manipulation interfaces. These would include both procedural and non-procedural approaches. The second requirement, and the one I wish to describe in detail, deals with the requirement for data independence.

What is data independence? For some of you this may be a new term. For others, let me give you my understanding. Data independence is the end result of a mechanism which permits two seemingly contradictory conditions to coexist. The first is that the data structure, as declared and stored in the database, is allowed to continue evolving, to support the enterprise's own evolution and to support the evolution of the applications systems running against the database. This evolution is permitted while simultaneously maintaining the operability of all the existing application programs which were written, tested, and put into production sometime in the past. All that was done when the database upon which they were running was quite different in its structure and optimization. The alternative, which we consider to be undesirable and actually not feasible, is that every production program must be reviewed for each projected database change. Each program affected would have to be modified, recompiled and retested. Then the database restructuring could be carried out. This alternative is so cumbersome, expensive and error-prone that the net result has been database stagnation with information systems effectively halted on some plateau in their development.

The DBTG specifications, as now published by the CODASYL Data Description Language Committee, provides a partial solution to this problem. Each application program knows the database through its subschema data description. If the schema itself is modified in a way that the existing subschemata remain a proper subset, then the matter is one of modifying the mapping statements within the subschemata to make them again conform. This is illustrated by Figure 1 which shows a number of subschemata on the right side and two schemata on the left. The lines between represent the mapping statements that tie the subschemata back to their source within the data declarations of the schema. The addition of new field and set types to existing record types, and the addition of new record types are handled quite easily as there are no existing maps which reference them. However, when records are split or combined, or when record, field or set implementation technique is changed, then the existing maps are placed under heavy stress and many must be changed. The SPARC Study Group is recommending that the DBTG schema be replaced with two new schemas. These are called the Internal Database Schema and the Conceptual Database Schema. In addition, SPARC defines an External Database Schema which is equivalent to the DBTG subschema.

The conceptual schema is considered to hold the best

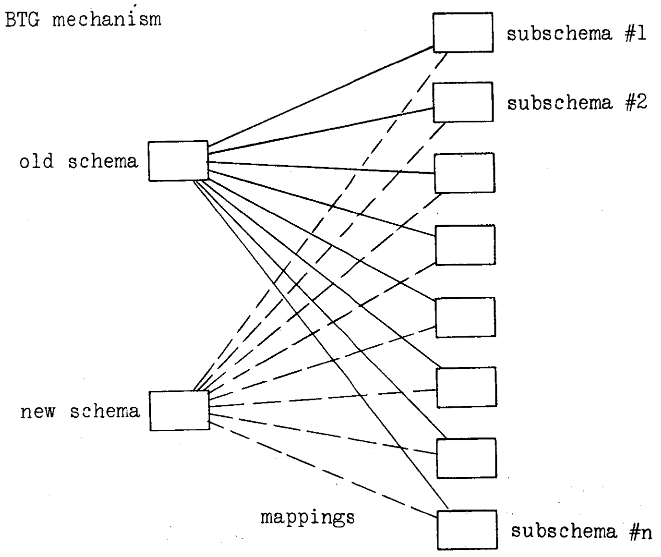


Figure 1—DBTG mechanism

available definition of the basic information of the enterprise. It would include record, field, and data-structure-set declarations representing real world entities and their properties, whether or not they have become a matter of immediate interest to the actual computer processing of data.

The creation and maintenance of the conceptual database schema is considered to be a business operations' responsibility rather than an informations systems operations' responsibility and administratively, would report to or be part of the enterprise's operations organization. A person, called the Enterprise Administrator, is assigned the responsibility for its creation and maintenance.

The internal database schema is the province of a person called the Database Administrator. He is charged with optimizing the performance, response time and media space requirement tradeoffs of the database system. He can change these optimization decisions from time to time without the requirement to modify, recompile or retest existing application programs.

Figure 2 illustrates the three kinds of schemas in the SPARC approach of things and the mapping between them. It should be contrasted with the DBTG approach illustrated in Figure 1.

The figure shows the internal database schemata on the left, the conceptual database schema in the center, and the multiple external databases schemata on the right. Now only the internal schema is changed to recognize new access or performance requirements.

The only mapping changes are in the one set of mappings between the internal and conceptual schemata. This advantage is really greater than is immediately obvious. One reason is that all of the performance, space, and response optimization aspects are concentrated in the internal schema. This permits the conceptual schema itself to take on a more stable and enlightened posture. This division of conceptual structure from storage structure is already observable in actions of the CODASYL Data Description Language Committee. Their publication of the Data Description Language Journal of Development[2] has already discarded certain storage description elements of the schema which had been specified by the Data Base Task Group Report.[3]

The list below suggests some of the optimization areas in which the Database Administrator could practice his skills:

- record implementation technique
- field implementation technique
- set implementation technique
- pointer implementation technique
- index implementation technique
- record placement statements
- file placement statements
- redundant data storage statements

For each conceptual record in the conceptual schema there would be a corresponding internal schema declaration which answers the following questions: (1) Is the record to be stored at all? (2) If it is to be stored, will it appear as a single linear record? (3) Will it appear as several separate linear records each with its own fields and storage rules. (4) Or, should it appear as a node in an inverted file structure? (5) If it is to be a linear record, should it be represented as a free standing database record, or (6) as a repeating group within a higher level database record? The separation of the conceptual record description from the internal record description has given the database administrator six choices, whereas the DBTG schema seems to limit him to one. Furthermore, it offers the opportunity to switch between them without obsoleting the external application programs or the maps that tie the external record descriptions back to the conceptual record descriptions. There are similar optimization declarations within the internal database schema with regard to implementation of fields, data-structure-sets, and indices. These

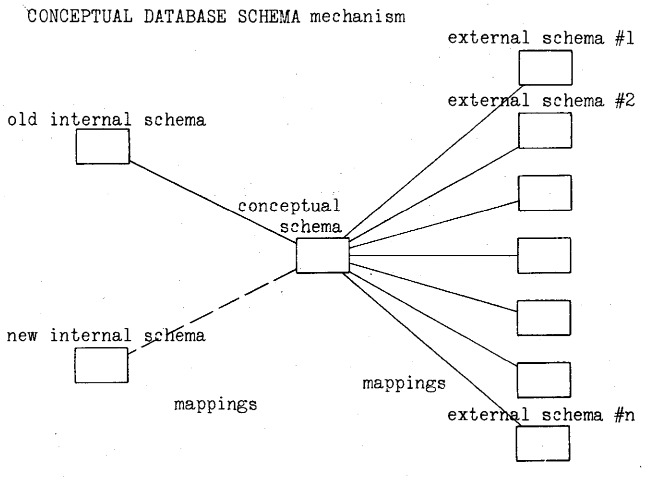


Figure 2—Conceptual database schema mechanism

can be changed without the collapse of programs interfacing at the external database manipulation level.

In support of these changes in the internal database schema, there must be database utility programs which can transform the database from its old internal format to its new internal format efficiently. Both batch *and* incremental restructuring utilities must be available.

The previous discussion treated with the evolution of the stored data, as described by the "internal database schema," to maintain efficient database operations. This was the province of the database administrator. Paralleling these changes, there also must be facilities by which the enterprise can change its own view of itself as represented by the conceptual database schema. These changes may result from structural changes in the enterprise, from changing views of the same structure or simply because the enterprise had been viewed narrowly and new information processing requirements have forced an expansion of viewpoint. The enterprise administrator is the person responsible for such changes. Many of these changes can be made without the loss of integrity to existing external database schemata. The anticipated result of most revisions to the conceptual schema is the phenomenon called "attribute migration" or "entity splitting." I can best explain this phenomenon through the use of an example that we discussed at a recent meeting. In this example, the conceptual data structure goes through a series of changes as the business information is better understood and new processing requirements are determined. Figure 3 contains two data structure diagrams.[4] The one to the left illustrates the conceptual data structure declared by the Enterprise Administrator.

This data structure diagram illustrates two conceptual record types, "company" and "personnel", with a data-structure-set "a" representing the fact that each person is associated with exactly one company and that each company has a set of personnel.

The right side of the diagram is supposed to suggest that there are a number of external database schemata as declared by the Application System Administrators. Schema #1, the only one fully visible, is a very simple, "flat file", view of the database. An application program accessing the database through this external schema would find a file of "pay-rec" records.

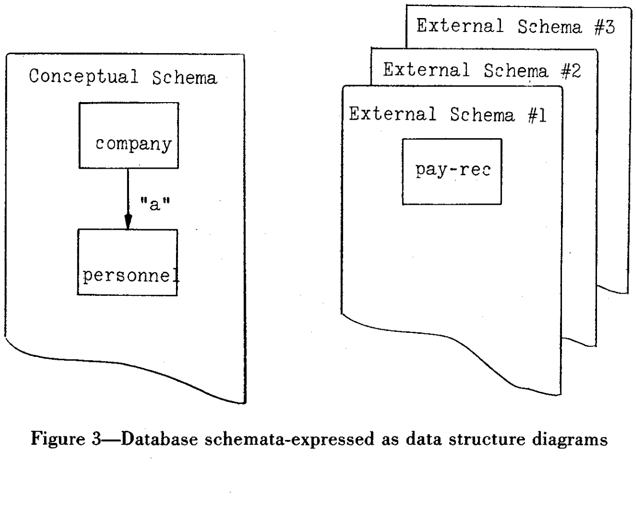


Figure 3—Database schemata-expressed as data structure diagrams

```
01 personnel.
  02 name.
  02 year-to-date-earnings
  02 employee-number PRIMARY KEY.
  02 office-phone.
98 MEMBER "a" SET, SELECT UNIQUE OWNER,
   DUPLICATES NOT ALLOWED.
```

Figure 4—Conceptual database schema

The conceptual database schema declaration of the "personnel" record type is illustrated in Figure 4. It should be noted that its field descriptions contain no declarations as to the type, length or mode of either the storage format of the fields or the format in which they should be delivered to a program which is accessing an instance of such a record. Note the I-D-S like 98 level entry which declares that the "personnel" record type is a MEMBER of the "a" data-structure-set type. It includes the declarations "SELECT UNIQUE OWNER," "DUPLICATES NOT ALLOWED" which relate to the integrity of the database. There are no set ordering declarations, as ordering is a matter of interest to a particular procedural program and might be specified differently in each external schema. If the set were declared to be implemented in the internal database schema, then its ordering rules would be specified for that purpose in an internal set declaration of the internal schema.

Note that this description is expressed in I-D-S/COBOLese modified to convey the new concepts. All of the reserved words are written in capital letters, and the variable names are in lower case letters. Please accept this for illustration purposes only. The SPARC study group is not charged with inventing languages and will deny any responsibility for the syntax I invented for this example.

The text of the external database schema #1 would look something like that illustrated in Figure 5. The "FD" is supposed to represent an external file declaration which would hold all the personnel records for the company with that company's code equal to "12498." Sequence is not declared in this example but would be necessary unless some arbitrary sequence were acceptable.

```
FD personnel-file, WHERE company-code EQUALS 12498
   01 pay-rec, SOURCE IS personnel
      02 name X(16)
      02 office-phone 9(10)
      02 year-to-date-earnings PIC(99,999.99).
```

Figure 5—External database schema #1

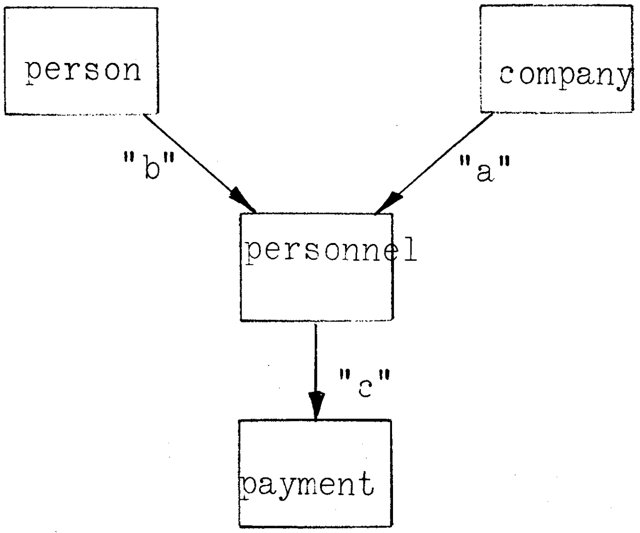


Figure 6—Modified conceptual database schema

The 01 entry record declaration source statement, "SOURCE IS personnel", is the most interesting aspect of this declaration. This is the mapping statement that relates the "pay-rec" record type in the external schema back to a "personnel" record type in the conceptual schema. Presumably, if the names were identical, then the source statement could have been left out and the default would be to assume the mapping to a conceptual record type with the same name. In our example, the 02 entries declaring the fields have no source statement and do make use of the default assumption in that they match with similarly named fields in the conceptual record type "personnel."

Given the initial state of the conceptual schema as set forth in Figures 3 and 4, let us assume that the following changes were made to the conceptual database declarations, while remembering that the existing external database schemata must maintain their effectiveness. There were two changes. First, it was recognized that the detailed earnings of each employee were of interest from time to time. Therefore, a new conceptual record type is declared for the "payment" record. It has a field called "earnings". The second change was to recognize that the personnel of the company were persons in their own right. In fact, it was discovered at the merger of several companies that some of the persons held two jobs and were personnel to two of the merged companies. The question now is how to change the personnel record declaration in the conceptual database schema to recognize these changes without obsoleting the existing external schemata.

Figure 6 illustrates the data structure diagram for the new conceptual schema. The data description of the conceptual schema in Figure 7 points out how these changes were handled. Note that there are two new 98 entries declaring the "personnel" record type's new relationships with the person" and the "payment" record types. More interesting are the modified declarations of "name" and "year-to-date earnings." The 02 entry for "name" has a source statement, "SOURCE IS OWNER OF "b" SET." In effect, the "name" attribute has migrated to the "person" record type, or more realistically the old "personnel" record type has been split into two record types, "personnel" and "person." The "name" followed the "person" aspect and the "office-phone" stayed with the "personnel" record. The 02 entry for "year-to-date-earnings" represents an even more interesting case of migration or split. Previously, the only data declaration was that of total earnings. Now we have both individual "earnings" in the "payment" record type and the old "year-to-date-earnings" in the "personnel" record type. The "year-to-date-earnings" has become a derived database object. Consequently, there is a new result statement associated with the "personnel" record type, "RESULT IS SUM OF earnings OF "c" SET WHERE year OF payment EQUALS year OF current-date." This is stating that the system is to compute this value based upon the summarization rules specified.

If you think of these changes in terms of the internal database schema, a number of questions arise. Did the Database Administrator actually implement these new conceptual record types with the corresponding internal record types. Assuming that he did so, then the further questions can be asked. Is the person's "name" to be redundantly stored in the "personnel" record? Or is it to be extracted from the "person" record whenever it is needed? This would be controlled by an internal database schema declaration. Similarly, is the "year-to-date-earnings" field actually to be stored in the "personnel" record or is it "virtual" and recomputed when needed? Clearly these are choices which the Database Administrator should have open to him without that choice having any logical effect upon the existing application programs.

The "ACTUAL" and "VIRTUAL" attributes in the DBTG schema illustrate the current state of the art. With the SPARC study group's separation of the storage aspects from the conceptual schema, the internal schema would show the fields only if they were to be actually stored.

Another efficiency question is whether the "payment" record type should be implemented as internal database records in a data-structure-set, or as a repeating group (array) within the "personnel" internal database record. The OCCURS clause in the DBTG schema is the means by which this storage issue is declared. Under the SPARC

```
01 personnel.
   02 name, SOURCE IS OWNER OF "b" SET.
   02 year-to-date-earnings, RESULT IS SUM OF earnings
      of "C" SET, WHERE year OF payment EQUALS year
      OF current date.
   02 employee-number, PRIMARY KEY.
   02 office-phone.
   98 MEMBER "a" SET, SELECT UNIQUE OWNER,
      DUPLICATES NOT ALLOWED.
   98 MEMBER "b" SET, SELECT UNIQUE OWNER.
   98 OWNER "c" SET.
```

Figure 7—Modified conceptual database schema

approach, the conceptual database schema does not recognize the concept of repeating group as it is considered a special and limited case of a record type within a data-structure-set. However, both the internal and external schemas would allow for their existence as they are useful in the storage and access of data and the programming of applications.

Let me take this evolution of the conceptual database structure forward two more steps. The first of these steps came about when the Enterprise Administrator decided to factor the address of residence out of the person record. For simplicity, let us assume that an address uniquely represented a place of dwelling. In other words, all persons with the same address were assumed to have the same residence. Figure 8 illustrates the addition of the "place" conceptual record type and the data-structure-set type "d" which associates occurrences of a "person" conceptual record type with a "place" conceptual record occurrence. The "address" in the "person" record will now migrate from the "person" record to the "place" record and the change is described with a source statement in the "person" record declaration.

Stretching the example the last step, it is now recognized that people move from place to place and that it is desirable to know current address as well as past addresses. Figure 9 illustrates the change in the nature of relationship between the "person" and the "place" records. Where there had been a n:1 relationship, there is now a n:m relationship. The new "address" conceptual record type serves to define this relationship. A result statement,

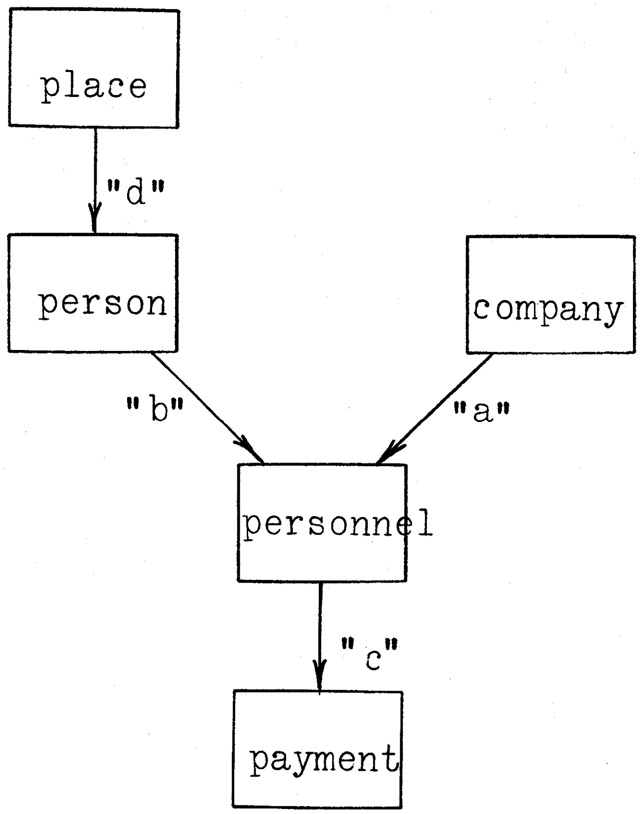


Figure 8—Further modification conceptual database schema

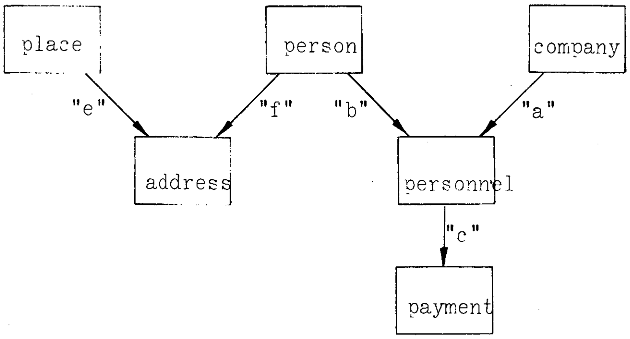


Figure 9—Some more modification conceptual database schema

02 address RESULT IS address OF THE FIRST MEMBER OF "f" SET

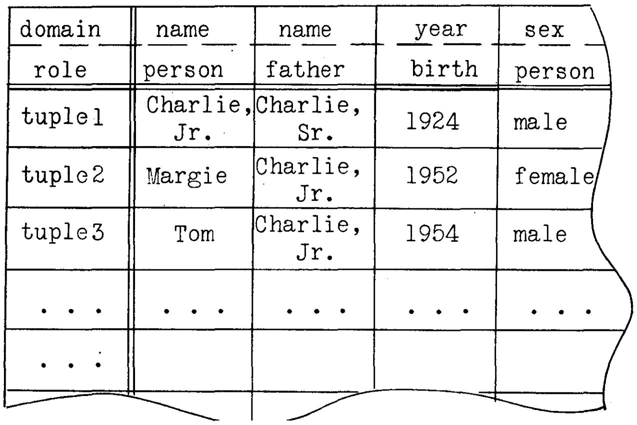
would be associated with the "address" field description in the "person" record description of the modified conceptual data base schema. In addition, the source statement,

02 address SOURCE IS OWNER OF "e" SET

would be associated with the address field description in the address record description.

These two statements jointly define the derivation function for the "address" field of the "person" conceptual record. Please accept the language as having been just invented to support the example. It is the intent of these statements to find the first "address" record and from that "address" record to find its owner "place" record via set "e" to obtain the specific address value. Some very interesting questions can be asked concerning the meaning of modifying a person's address. Does this mean to create a new "address" record relating the person to a new place and further to create a new "place" record for the designated address if one does not currently exist? Or, does it mean to change the address of the place and all the other persons who currently or once lived there? Clearly, I believe this means the former. However, modifying the "address" field of a "place" record would mean that the dwelling has somehow had its address changed to correct an error, or perhaps all the houses on that street had been renumbered in some unification plan. This happened to me once when an entire section of town was renumbered.

I would like to shift the subject now to another very important area. This is an area where there is considerable technical debate at the present time. A formal debate was held at the ACM SIGFIDET meeting a year ago. This is the debate as to whether the *Relational Model* or the *Data-Structure-Set Model* is the best one upon which to build future database management systems. Codd of IBM



domain	name	name	year	sex
role	person	father	birth	person
tuple1	Charlie, Jr.	Charlie, Sr.	1924	male
tuple2	Margie	Charlie, Jr.	1952	female
tuple3	Tom	Charlie, Jr.	1954	male
. . .	. . .	. . .	. . .	. . .
. . .				

Figure 10—Relational model example

Research is the author and a very articulate sponsor of the relational model.[5,6] Date of IBM (Hursley) has also written extensively[7] on the subject. It would be presumptous for me to try to explain the position of the supporters of the Relational Model in the limited space available. However, it is fair to say that the relational model has received tremendous support within the academic circles in the United States. One of the reasons is its use of set theoretic operations. This makes it very attractive to mathematically trained people. Another factor is its promise of usefulness in handling enquiries as a non-procedural language.

The competitive model in the current debate is the data-structure-set[8] or network model. This is the model that underlies both DBTG systems and Honeywell's Integrated Data Store. This is the model that I developed and have supported for a number of years. This model is so well established that it was described, in the ACM Communications news article reporting the debate, as being the "establishment" point of view. The most critical question for all of us is, "What is the significance of the debate?" Is it "full of sound and fury signifying nothing" or does it portend a radical change ahead? Let me oversimplify Codd's position as follows: "The data-structure-set model is based upon an access mechanism and has no place in a high level language approach to database systems. It will ultimately be replaced and disappear." I take a different approach to the subject. I consider data-structure-sets as representing natural relationships which exist in the real world. These are the relationships between teachers and students, between companies and customers, or as in the earlier example between persons and places. For information processing purposes, these relationships can be recorded by carrying around redundant data (the relational model) or by one of the many set implementation techniques.[9] Possibly the redundant data fields used in the relational model should just be characterized as the 10th set implementation technique and closely related to the "phantom" technique.

This might be the ideal set implementation technique if and when hardware support for associative retrieval is achieved and becomes competitive with directly addressed secondary storage.

I consider the relational model and the data-structure-set model essentially compatible and subject to transformation from one form to the other. In the months since the debate last May, I have been attempting to clarify this transformability between the models. During this time, a third model, the ***Data Independence Access Model***[10] (DIAM) has also been studied for comparison. It was developed at IBM Research by Senko and his associates. It has been found to offer some interesting perspective upon the debate. The thing that is most interesting is that the relational model and the data independence access models appear to be completely at odds with each other. At best, they are at the opposite ends of a spectrum. By comparison, the data-structure-set model seems to be an effective hybrid between these two extremes.

Can this amalgam be explained in a few words? Let me try. The relational model, at first glance, appears to be concerned with only fields and records. Of course, it is the way in which one uses the fields that makes the system work.

Figure 10 illustrates the use of the relational model to describe certain aspects of information about a person. Each row is called a "tuple" and is similar to a record. Each column is a "role" on a "domain" and is similar to a field.

The DIAM model seems to work only with single field records and the 1:1 directed relationships that exist between these single field records. Fields are only used for the unique or primary key identification of a record. The same information about people used in Figure 10 for the relational model is recast in Figure 11 for the DIAM model. The ovals represent entities and their unique identification. The arrows between the ovals represent directed 1:1 relationships between a pair of such entities.

The data-structure-set model, as implemented by DBTG

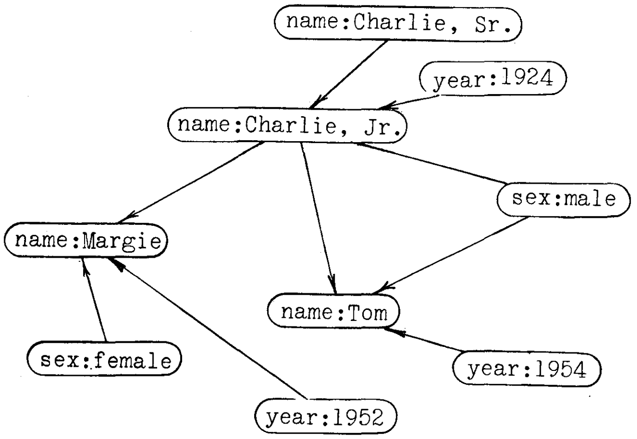


Figure 11—Data Independence Access Model example

and I-D-S, offers the database designer the choice of whether to represent a property of an entity as a field or as a data-structure-set according to which approach seems more natural to him. Figure 12 recasts the information used in the last two figures into the data-structure-set model. In this case, the blocks represent records and their associated data fields. The multi-headed arrows represent the 1:n relationships between a record in an "owner" role and the records in the "member" role of a data-base-set.

In the example I have chosen to model "year of birth" and "sex" attributes as fields and the "parent:child" relationship as a data-structure-set.

In the prior example describing the conceptual database schema, the splitting of entities and the resultant migration of attributes was the prominent action described. What actually happened was that the Enterprise Administrator progressively shifted his viewpoint from a mostly relational model viewpoint toward a more DIAM-like viewpoint. However, during this entire time, the conceptual schema was always a mixture of fields, records and data-structure-sets. It was always a data-structure-set model type of compromise.

The third area which I want to discuss deals with the use of hardware and firmware to assist in handling time consuming operations which appear frequently in database systems. While progress in this direction is limited, a trend is suggested. The virtual memory hardware pioneered by the Ferranti Atlas and the Honeywell MULTICS systems provided hardware support for addressing a paged database. The IBM 370 has followed this course. While the address space of these machines is too limited in size for most databases, they do represent a significant step forward. Time should take care of the size problem.

A second example of hardware and firmware support was announced last year for the Honeywell Series 60 Level 64.[11] This support focuses upon the data independence issue discussed earlier. The most time consuming aspects of supporting data independence is the need to continually transform data from the stored format (internal schema) to the format in which the program expects it (external schema). This requires a field by field examination and transfer. The H60/64 has provided a rudimentary set of field instructions which are controlled by data descriptors. These are the "MOVE A TO B" and "COMPARE A WITH B" instructions. These instructions operate through a pair of field descriptors. Each descriptor identifies the offset of the field within a record or within the program's work space or stack frame and identifies a base register which establishes the record's, program's or stack frame's location within the processes address space. It also states the field's length and data type. (BCD, ASCII, EBCDIC, packed decimals, . . .). Thus during a move with type control, all required conversions are made and the result is truncated or filled with blanks or zeros according to the description of the receiving field.

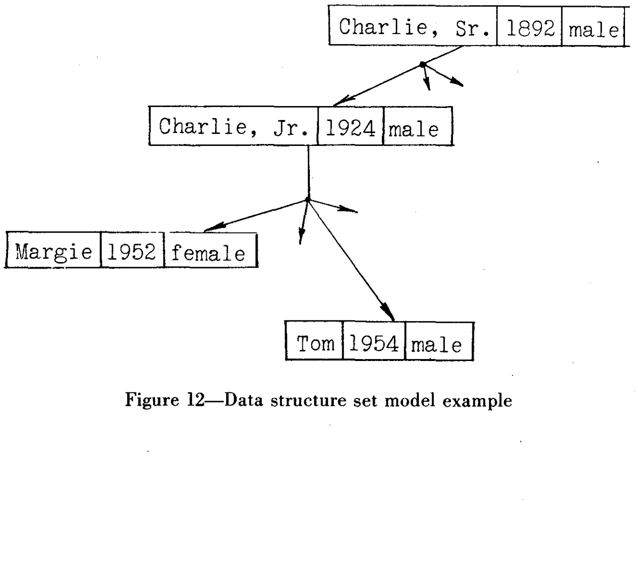


Figure 12—Data structure set model example

The H60/64 also has a "hashing" or randomizing command in the instruction set. This instruction is designed to operate with fields of various lengths and with multiple fields which are not contiguous. The instruction is based upon a folding, shifting, negating algorithm which is particularly sensitive to the fact that identifiers, especially numeric, stored as character strings do not make use of the high order bit patterns. The algorithm is patterned after the I-D-S randomizing routine which has more than ten years of field exposure on three different machine architectures and which has almost universal acceptance by I-D-S users on those machines.

The selection of these three commands to be supported by hardware was based upon instrumentation of actual I-D-S applications on the H6000 line. The instrumentation determined which software routines would yield the greatest return if reimplemented in hardware.

In summary, the art of database management is evolving and has made significant progress. The DBTG type systems pioneered by the Integrated Data Store have become available from a number of computer manufacturers. The evolution of these DBTG systems to the ANSI/SPARC tripartite data descriptions is under way and should continue. The current debate on data-structure-set vs. relational models of data will find its greatest contribution in the development of a greater understanding of the nature of data. This is a little like the continued study of particle physics; it is a seemingly unending struggle for fine and finer resolution. The introduction of database commands into the instruction sets of our computer is a yet unproven aspect. Unproven in that their motivation is one of efficiency. If the operations which are assisted by this hardware are executed with sufficient frequency, then they pay off. If the frequency diminishes, then the hardware investment is lost. It would now seem that money invested in "scatter/gather" instructions and formatted track discs to support unit record processing at the physical I/O transfer level may have been wasted as their utilization is very low in the new database systems.[12] However, if some advances are false, still others are true and pay their way. It appears to this author that there is still a good yield to be gained in providing new hardware assistance in support of well defined database functions.

REFERENCES

1. Bachman, C. W., *Summary of Current Work*—ANSI/X3/SPARC/Study Group—Database Systems revised January 29, 1974.
2. "CODASYL Data Description Language," *Journal of Development*, June 1973, NBS Handbook 113, U. S. Department of Commerce.
3. CODASYL Data Base Task Group Report, April 1971 (various sources).
4. Bachman, C. W., "Data Structure Diagrams," *Data Base* 1, 2, Summer 1969, *Quarterly Newsletter of ACM SIGBDP*.
5. Codd, E. F., "A Relational Model of Data for Large Shared Databanks," *Comm. ACM 13*, pp. 377–387, June 1970.
6. Codd, E. F., "A Database Sublanguage Founded on the Relational Calculus," *Proceedings of the 1971 ACM SIGFIDET Workshop*, pp. 35–68.
7. Date, C. J., "Relational Database Systems: a Tutorial," 4th International Symposium on Computer and Information Science (COINS-72).
8. Bachman, C. W., "Set Concepts for Data Structures," *Encyclopedia of Computer Science*, Auerbach Corp.
9. Bachman, C. W., "Implementation Techniques for Data Structure Sets," *Data Base Management Systems*, D. A. Jardine (editor), North-Holland Publishing Company, 1974, pp. 147–160.
10. Senko, M. E., E. G. Altman, M. M. Ashtrahan, P. L. Fehder, "Data Structures and Accessing in Data-Base Systems," *IBM Systems Journal*, #1, 1973.
11. Atkinson, T., "Architecture of Series 60/Level 64," *Honeywell Computer Journal*, Volume 8, No. 2.
12. Buzen, J. P., "I/O Subsystem Architecture," *Proceedings of the IEEE*, Vol. 63, No. 6, June 1975,

A data sorting system using a high speed bus

by P. M. THOMPSON and Z. H. GLANZ
University of Ottawa
Ottawa, Canada

SUMMARY

The access time for content addressed data can be significantly improved if it is filed according to an appropriate set of descriptors. A further improvement can be obtained if hardware, rather than software, techniques are used to implement the file or sorting memory. The paper discusses the design of a hardware implemented system which can sort and re-sort digital data according to any set of descriptors chosen. The main limitation to the speed of sorting and re-sorting is the intercommunication system between the various parts of the store. The application of segmented ring bus techniques gives a high sorting speed and a flexible system. The memory design is suitable for LSI because it consists of a number of similar units capable of integration using silicon technology.

INTRODUCTION

In general, digital data storage systems can be classified under the following headings:

(a) Direct addressed storage
(b) Associative memories
(c) Sorting arrays

Much has been published about the first two and this paper is concerned mainly with the latter. A sorting array has some of the characteristics of an associative memory, but the position of an entry in the array depends upon a characteristic of the entry: e.g., several descriptor bits masked positionally. For i descriptor bits, there is a maximum of 2^i sets of entries. A characteristic of a sorting array is that the entries are sorted into new sets if the position of the mask for the descriptor bits is changed. An example of the process is that this paper might be filed under one of several index terms and the choice of a new index term would result in a change of file. In many applications, the speed with which data entries can be resorted is important and software techniques prove to be slow.

It follows that there is a need for a hardware implemented sorting system. Kautz[1] designed a Logic-in-Memory system to deal with the sorting of small amounts of data, but this approach unmodified becomes cumbersome as the system grows. Associative memories can also be used within a sorting system, but there are difficulties associated with the communication between the separate associative memory units. Indeed, it seems that the main limitation to the speed with which a re-sorting operation can be performed is the flexibility and speed of the intercommunication system between the different storage units in the overall system.

SEGMENTED BUS

A solution to the intercommunication problem is provided by the segmented bus.[2] For a sorting array, it is used in its simplest form where it is similar to a parallel shift register one word wide with various inputs and outputs along its length. Words are transmitted by being clocked from segment to segment along the bus in "carriers". Words can be entered at input/output segments, or ports, if there is an empty carrier. It is possible to enter several words at the same time if there are empty carriers at the appropriate ports. Similarly, several words can be output at the same time. When a word is output its place can be taken by a new input word. Thus the segmented bus provides communication between a pair of ports without preventing communication between another pair, and it can provide communication between several pairs at the same time. It is usual to clock the whole system, bus and input/output units, at the same speed.

SIMPLE RING CONFIGURATION

A simple arrangement for a sorting memory is shown in Figure 1. A segment of the bus is extended to become a memory segment, which consists of a memory unit and compare and control logic associated with the appropriate segment of the bus. An additional segment is used as an input/output unit.

This ring can perform three basic functions: (a) Input information can be clocked around the ring until it reaches an appropriate memory segment, where it is stored; (b) A re-sorting operation may be performed by changing the descriptors for the different memory units; (c) Words may be output on an associative basis. In Figure 1, the memory units are first-in-first-out (FIFO) stacks and the control logic is routing logic, which can perform the following operations: (a) Words arriving from the left are routed to the top of the stack if they contain the appropriate descriptor for that stack; (b) Words that do not contain the appropriate descriptor are routed straight through to the right; (c) Whenever an empty carrier enters the segment, a word at the

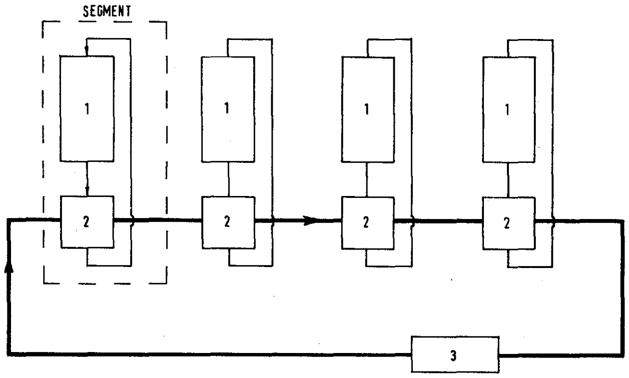


Figure 1—Sorting ring: (1) Memory unit, (2) Compare and control logic, (3) I/O and function control logic

bottom of the stack can be routed to the right if it does not contain the appropriate descriptor for the stack; (d) Words at the bottom of the stack that contain the appropriate descriptor are routed straight through and returned to the top of the stack.

COMPLEX CONFIGURATIONS

The number of memory units on a ring grows with the number of words to be stored and descriptors used. However, although the segmented character of the bus allows multiple entry and exit of data words, the single channel characteristic of the ring limits the information flow and results in extended sorting and re-sorting times.

The speed of sorting and re-sorting can be improved by complex bus configurations. A good criterion for comparison of these configurations is the maximum re-sorting time because this is a well defined parameter and always larger than the initial sorting time. For a simple ring bus, the maximum re-sorting time is given by:

$$T_{(\max)} = W_m \cdot M_n \cdot t_c$$

where: $T_{(\max)}$: maximum re-sorting time
W_m: No. of words in a memory unit
M_n: the total No. of memory units
t_c: clock period.

If a multiple ring configuration such as that shown in Figure 2 is employed, then:

$$T_{(\max)} = K \cdot W_m M_r t_c = K \cdot W_m \cdot (M_n/R_n) \cdot t_c$$

where $K > 1$

and R_n: the total No. of rings.
M_r: No. of memory units on a ring.

The improvement in sorting time is: R_n/K.

Values for K were determined by the simulation of some single ring and double ring configurations. Values chosen for M_n were 16, 32 and 64 and values chosen for M_r were 4 and 8. A random selection of words was first stored in the system and sorted according to one descriptor set. Then the descriptor bits were changed and the number of clock pulses for the completion of the re-sorting process was determined. K had an average value of approximately 1.7 and a worst value of approximately 2.

The memory system can be expanded by coupling the ring shown in Figure 2 into a further ring segmented bus. This further ring can be coupled into another and so on depending upon the size of the overall system desired. In large systems, the designer has the choice of either employing a large number of descriptor bits and increasing the word length, or using the same descriptor for several memory segments. The rings are coupled by "ring control segments" which function as part of a major ring much as do the memory segments in a minor ring. The control logic is also similar, except that there are three inputs and three outputs, rather than two of each as in a minor ring. Thus, in an integrated realization, ring control segments can be used as memory segments with one input and one output left unused. As in the case of the memory control logic, the ring control logic can route the data from any input to any output as determined by the descriptors.

MEMORY SEGMENT

Two types of memory segment have been designed, an elementary static logic unit and a more flexible dynamic unit where the storage can be in a dynamic MOS shift register or a disk. Both units can be used as either a memory segment or a ring control segment, depending upon the number of bus inputs and outputs connected. The static logic unit is shown in Figure 3. It consists of two static registers (block No 1), the one on the left an output register and the one on the

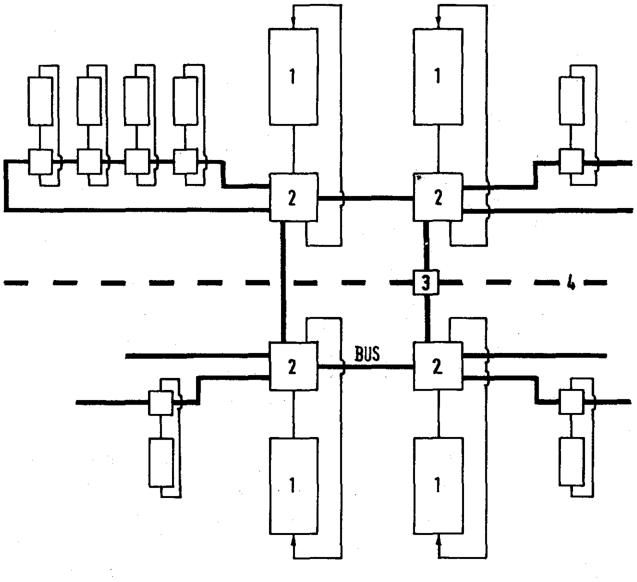


Figure 2—Four ring sorting system (1) Aux. register, (2) Ring control logic, (3) I/O and function control logic, (4) Bus for more complicated system

right an input register. A compare and control unit in block 2 communicates between the registers and the bus. It allows words containing the descriptor for the segment to enter the input register and outputs words from the output register into empty spaces on the bus. To control its next function, it also counts the words entering the input register. When the output register is empty, this is detected by the control logic and status information transmitted to "Central Control". When the resort operation is complete, the words in the input register can be shifted through block 3 into the output register under the control of the counter in block 2.

DYNAMIC MEMORY UNIT

The inefficient use of storage space in the static logic system of Figure 3 is overcome in the system of Figure 4. Here a large memory (block 1) is used with two auxiliary registers (block 5) which become the input and output registers. These auxiliary registers are small, static logic FIFO stack memories which are coupled to the bus through control logic in a similar way to the input and output registers of Figure 3. The main memory of block 1 can be any read-write memory, but is preferably a disk or a dynamic shift register. (In the description below the main memory is a dynamic shift register.)

An advantage of this type of memory unit is that it can become content or position addressable, according to control signals. The unit can thus be used in any of the types of memory system given in the introduction. Note the similarity between entering a word into a dynamic shift register and a bus. In both cases, new information is introduced into empty carriers or spaces in the stream and words with the appropriate descriptors are removed from the stream.

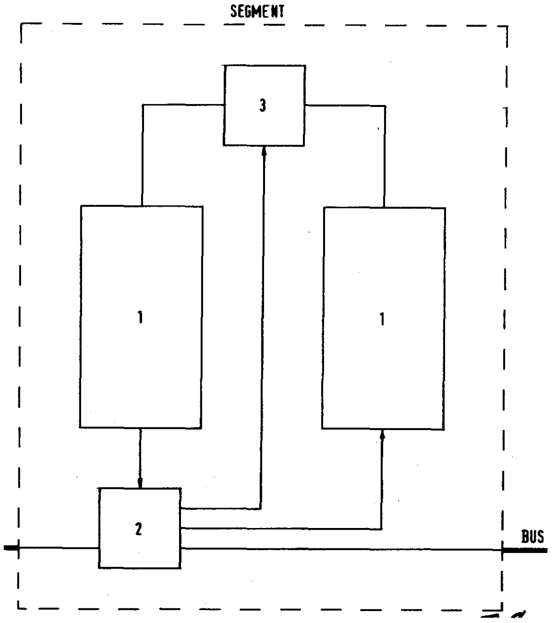


Figure 3—Static memory unit: (1) Static register, (2) Compare and control logic, (3) Logic

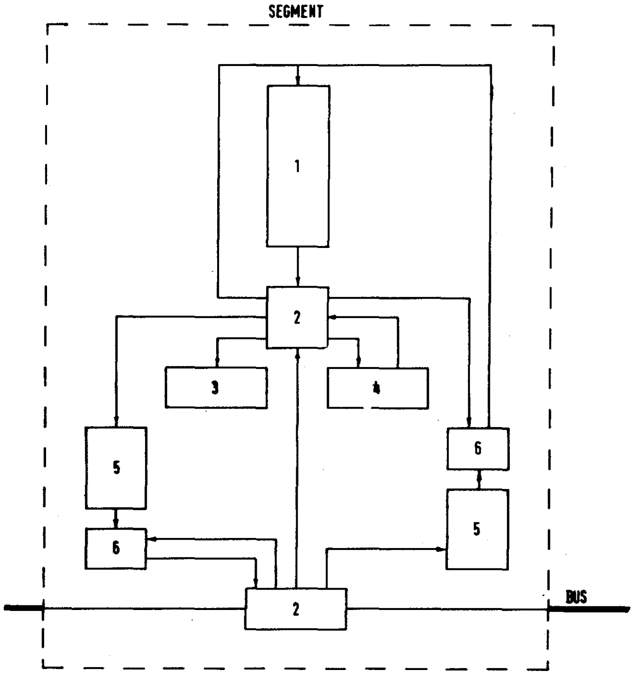


Figure 4—Dynamic memory unit: (1) Dynamic register, (2) Compare and control logic, (3) Word counter, (4) Cycle counter, (5) Aux. register, (6) Control logic

The dynamic memory unit operates in the following manner: During the first cycle of the main memory, counter B (block 4) will be incremented every time a word passing the control unit (upper block 2) does not match the descriptors for the segment. When the first cycle is finished, the number in counter B indicates how many words will leave this segment in the process of re-sorting. Whenever space is available in the output auxiliary register, words which do not match the descriptor for the segment will be transferred there. During this operation, there is another transfer operation; words arriving from the bus enter the input auxiliary register through the control unit (lower block 2) and are transferred into the empty spaces of the main memory.

CONCLUSIONS

It has been shown that a sorting memory, or digital data filing system, can be constructed from memory segments interconnected using systems of unidirectional segmented ring busses. Furthermore, the memory units within the memory segments can also take the form of ring structure. The control logic units used for coupling rings can be treated as coupling filters and the same type of coupling filters can be used throughout the system, with the differences being only in the choice of descriptor and descriptor position.

There are many applications for sorting arrays and these include business filing systems, data acquisition, many medical applications including diagnostics and library systems. In addition to this, they can be applied to software housekeeping, and the coupling filters might be used for routing in data-packet communication systems.

Modifications to the design of the memory segments permit them to be used as either content or position addressable systems, as determined by control inputs. Thus it is possible to design a system capable of operating in all the modes classified in the introduction.

ACKNOWLEDGMENTS

The authors would like to acknowledge the contribution of M. Kreiger for useful discussions and many good ideas, of K. Szabo and P. Charlebois for their work on the realization of memory modules and of Microsystems International Ltd. and General Instruments of Canada Ltd. for providing the integrated memory circuits used.

REFERENCES

1. Kautz, W. H., "Cellular Logic in Memory Array," *IEEE Transactions on Computers*, Vol. C-18, No. 8, August 1969.
2. Champagene, C., "A Bus Structure for a High Data Rate," submitted to *AFIPS Conference Proceedings* 1975.

INFOPLEX—Hierarchical decomposition of a large information management system using a microprocessor complex

by STUART E. MADNICK
Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

The effective use of computer systems for large-scale information management rather than numerical computation is still a largely unsolved problem. In this paper important concepts and theories regarding computer architectures for information management are introduced. The underlying concept, hierarchical decomposition, is presented in the next section.

Several ongoing developments in computer technology have made a radical change in system architecture both necessary and feasible.[21] New memory technologies, some recently announced and others under development for the near future, make very large capacity memories possible. But, the physical organization of such memories and their logical information handling functionalities are yet to be determined. As an example, consider the objective of designing an information system with total storage capacity in excess of 10^{15} bits processing up to 10^6 logical interactions (e.g., queries, updates) per second and capable of physical input/output rates of at least 10^9 to 10^{10} bits per second. No present computer architecture or theoretical structure has explicitly addressed the design of such a system.

In order to attain very high performance and functionality, as indicated above, it is necessary to take advantage of extensive parellelism in the system. With the advent of microprocessor technology such a strategy is quite feasible. Whereas highly parallel computer systems of the past, such as ILLIAC IV and CDC STAR-100, were designed to solve numerical problems, totally new approaches are needed for information management problems.

By using hierarchical decomposition, both functional and physical, a highly parallel information management system architecture can be implemented by means of a processor complex. Such a system, called the INFOPLEX, is presently under study at the Center for Information Systems Research in the MIT Sloan School of Management.

HIERARCHICAL DECOMPOSITION

There are two major types of parallelism that can be exploited in an information system: functional and physical.

Function decomposition

In almost all cases, the interactions with an information system are in terms of very high level concepts whether originating from a human at a terminal or another computer system. These requests must be converted into the more basic operations appropriate to the particulars of the physical hardware and information structures. There are many ways that this conversion can be accomplished but in our research[7] we have found the technique of *hierarchical function decomposition* to be very effective for advanced information management systems (similar techniques have been used successfully in operating systems[13] and basic file systems[11]).

Figure 1 outlines the hierarchical function decomposition used in our Generalized Management Information System (GMIS) effort. This hierarchical concept has been used in the implementation of the New England Energy Management Information System (NEEMIS).[7] Note that each functional level of the hierarchy is implemented in terms of the functions provided by the next lower level. This strictly hierarchical approach has resulted in an extremely powerful, flexible, and modular information management system—a primary requirement for the NEEMIS application. In addition, the conciseness of the structuring greatly reduces the complexity of the system making optimization and debugging much more effective.

Physical decomposition

To date, the technologies that lend themselves to low cost per byte storage devices (and, thereby, economical large capacity storage) result in relatively slow access times. If it were possible to produce ultra-fast limitless-

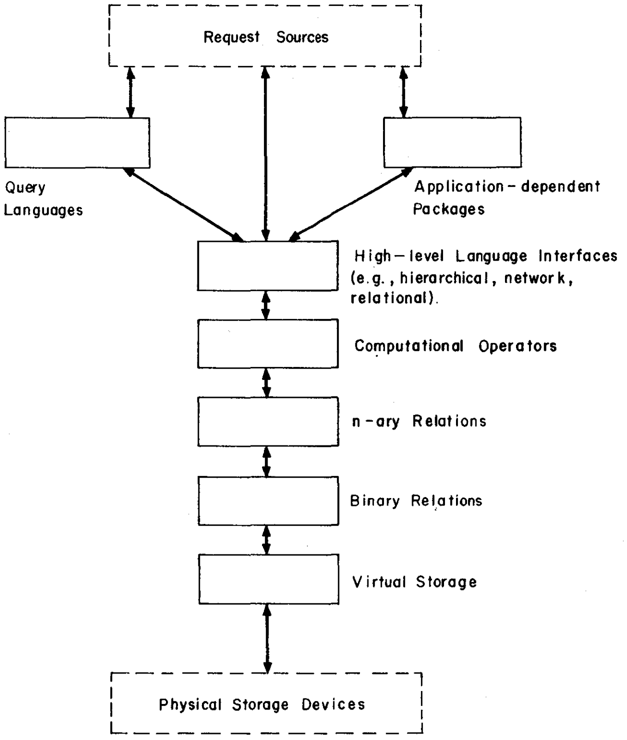


Figure 1—Hierarchical function decomposition

Storage Level	Random Access Time	Transfer Rate (bytes/second)	Cost per Byte	Technology
1. Cache	50 ns	100M	100¢	Semiconductor RAM
2. Main	1 μs	16M	10¢	Semiconductor RAM, Ferrite core
3. Block	50 μs	8M	2¢	Semiconductor shift registers, Bulk ferrite core, Charged-coupled devices, magnetic bubbles
4. Backing	1 ms	2M	.5¢	Fixed-head disks and drums, charge-coupled devices, magnetic bubbles
5. Secondary	50 ms	1M	.01¢	Moving-head disks
6. Mass	1 sec	1M	.0005¢	Automated tape-handlers, Laser devices

Figure 2—Spectrum of Storage Device Technologies

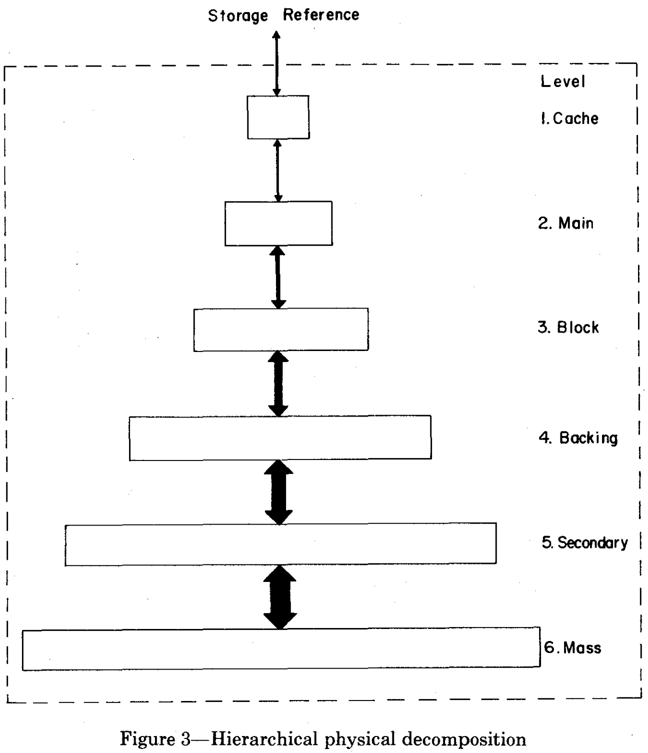


Figure 3—Hierarchical physical decomposition

capacity storage devices for miniscule cost, there would be little need for a physical decomposition of the storage. Lacking such a wonderous device, the requirements of high-performance yet low-cost are best satisfied by a mixture of technologies combining expensive high-performance devices with inexpensive lower-performance devices.

Figure 2 indicates the range of performance and cost for typical current-day storage technologies divided into six cost-performance levels. New storage technologies will undoubtedly make improvements at all levels, possibly even collapsing some. In any case, it does appear that multiple levels of cost-performance storage devices will continue to exist for many years to come.[15,22] Note in particular that the current spectrum of devices represented in Figure 2 span over six orders of magnitude in both cost and performance.

If all references to information in the system were random and unpredictable, there would be little utility for the intermediate levels of storage technologies. Most practical applications result in clustered references such that during any interval of time only a subset of the information is actually used, especially when you consider the use of indexes and other control information. This phenomenon is known as *locality of reference.*[6] Under such circumstances, the intermediate levels of storage technologies can be used as *paging devices* or *staging devices* that hold these information clusters. This approach has been used in contemporary systems on the microscopic level[5] (e.g., IBM System/370 Model 158 and 168 cache systems), intermediate level[2,3,8,9,16,17,18] (e.g., Honeywell 68/80 Multics paging system), and macroscopic level[4,10] (e.g., IBM 3850 Mass Store System).

There are many ways that such an ensemble of storage devices may be structured but in our research[12,14] we have found the technique of *hierarchical physical decomposition* to be very effective. A detailed explanation of this approach is presented in Reference 14. Briefly, information is moved between storage levels automatically depending upon actual or anticipated usage such that the information most likely to be referenced in the future is kept at the highest (fastest access) levels. Figure 3 depicts the general structure of a hierarchical storage system.

PARALLELISM IN HIERARCHICAL STRUCTURE

The original research in hierarchical function decomposition was motivated by the desire for more structured implementation; the research in hierarchical physical decomposition was motivated by the desire for improved system performance. In addition to these original benefits, the hierarchical structure also lends itself to considerable parallelism.

Asynchronous function decomposition

As noted earlier, each level of function decomposition is implemented in terms of the primatives of the next lower level (refer to Figure 1). Furthermore, usually several lower level primatives must be used to implement each higher level primative. By using separate sets of processors for each functional level, it is possible to take advantage of parallel execution of lower level primatives, specialize processor functionality, simplify implementation, and enhance modularity. Thus, a request for a lower level function is accomplished by an inter-processor signal to one of the processors that implement that level. The hierarchical structure of the function decomposition makes such inter-processor communication relatively simple and efficient.

By incorporating queueing facilities and internal multiprogramming within each of the processors, a high performance "pipeline" can be attained. This makes it possible to maintain high rates of throughput. Furthermore, the simplicity of the structure makes relatively unlimited modularity possible thereby making it possible to assemble systems of enormous performance capacity.

The multiple processor implementation of the hierarchical function decomposition is depicted in Figure 4. Although such extensive use of processors has been quite expensive in the past, the advent of low-cost microprocessors makes such a system economically feasible. Furthermore, since each level implements only a limited amount of the total system's functionality, very simple processors can be used.

By taking advantage of the particular structure of the system and the somewhat specialized functions that are performed, highly reliable operation can be attained using techniques such as those used in the PLURIBUS system.[19] One of the key properties of the hierarchical function decomposition implementation is that all processors are anonymous and act as interchangeable resources (within a function level). Thus, if a processor malfunctions or must be removed from service, the system can continue to function without interruption. After a reasonable amount of time has elapsed, the higher level processors that had generated requests that were being performed by the defective processor merely need to reissue the same requests. Alternatively, the reissuing of requests could be accomplished automatically by the inter-level request queue mechanism.

Although the details are not elaborated in this paper, it should be clear that extensive parallelism, throughput, and reliability can be attained by means of a multiple processor implementation of the hierarchical function decomposition.

Asynchronous physical decomposition

As this author has noted in Reference 14, it is possible to generalize the current-day specialized hierarchical storage systems (e.g., cache systems, paging systems, file archiving systems). In such a generalized system, the storage system is physically decomposed into a hierarchy of storage levels operating under local controls (i.e., decentralized control) with limited coordination necessary between levels.

Various physical storage management and movement

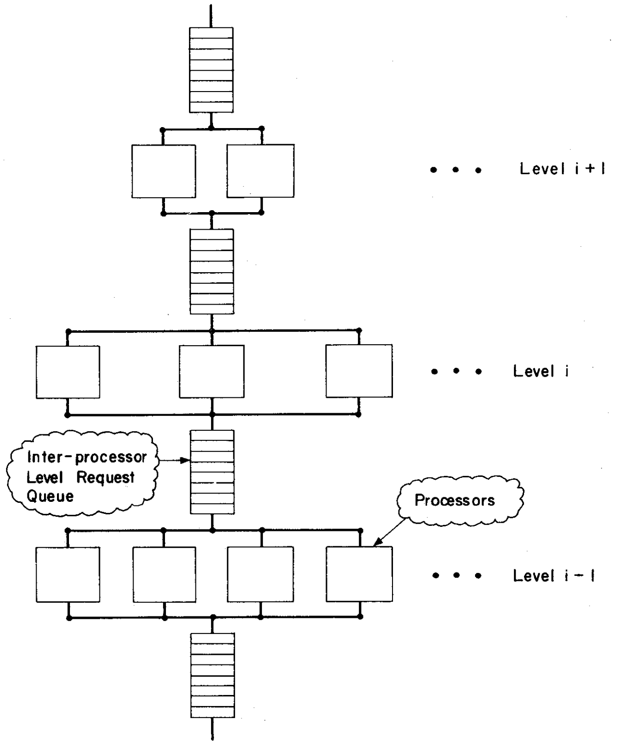


Figure 4—Hierarchical function decomposition using a microprocessor complex

techniques, such as page splitting, read through, and store behind, can be distributed into the hierarchy of levels. This facilitates parallel and asynchronous operation in the hierarchy. Furthermore, these approaches can lead to greatly increased reliability of operation. For example, under the *read through* strategy, when data currently stored at level i (and all lower performance levels $k>i$) is referenced, it is automatically and simultaneously copied and stored into all storage levels $j<i$ (i.e., all higher performance levels). The data itself is moved between levels in standard *transfer units,* also called *pages,* whose size $N(i\text{-}1,i)$ depends upon the storage level from which it is being moved. (See Figure 5 for an illustration of this process.) Since all upper storage levels receive this information simultaneously, if a storage level must be removed from the system, there are no changes needed. In this case, the information is "read through" the level as if it didn't exist. Since all data available at level i is also available at level $i+1$ (and all levels $j>i$), there is no information lost. Thus, no changes are needed to any of the other storage levels or the storage management algorithms although we would expect the performance to decrease as a result of the missing storage level. A limited form of this reliability strategy is employed in most current-day cache memory systems.[5]

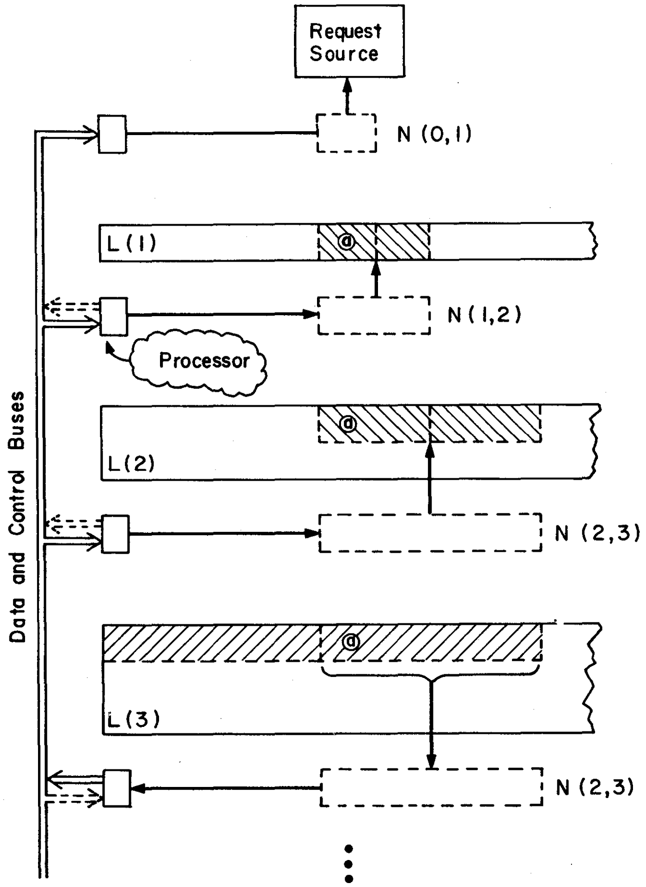


Figure 5—Example of read through with page splitting

In a *store behind* strategy, all modified information is initially stored in $L(1)$, the highest performance storage level. This information is marked "changed" and is copied into $L(2)$ as soon as possible, usually during a time when there is no activity between $L(1)$ and $L(2)$. At a later time, the information is copied from $L(2)$ to $L(3)$, etc. A variation on this strategy is used in the Multics Multilevel Paging Hierarchy.[8]

The store behind strategy can be used to provide high reliability in the storage system. Ordinarily, a changed page is not allowed to be purged from a storage level until the next lower level has been updated. This can be extended to require two levels of acknowledgment. For example, a changed page cannot be removed from $L(1)$ until the corresponding pages in both $L(2)$ and $L(3)$ have been updated. In this way, there will be at least two copies of each changed piece of information at levels $L(i)$ and $L(i+1)$ in the hierarchy. Other than delaying the time at which a page may be purged from a level, this approach should not adversely affect system performance. As a result of this technique, if any level malfunctions or must be serviced, it can be removed from the hierarchy without causing any information to be lost. There are two exceptions to this process, $L(1)$ and $L(n)$. To completely safeguard the reliability of the system, it may be necessary to store duplicate copies of information at these levels only. Figure 6 illustrates the two-level store behind algorithm.

A considerable amount of work is required to manage the storage hierarchy. This work can be distributed by means of multiple processors servicing separate physical storage levels. In this manner, these processors may operate asynchronously and in parallel. A simplified

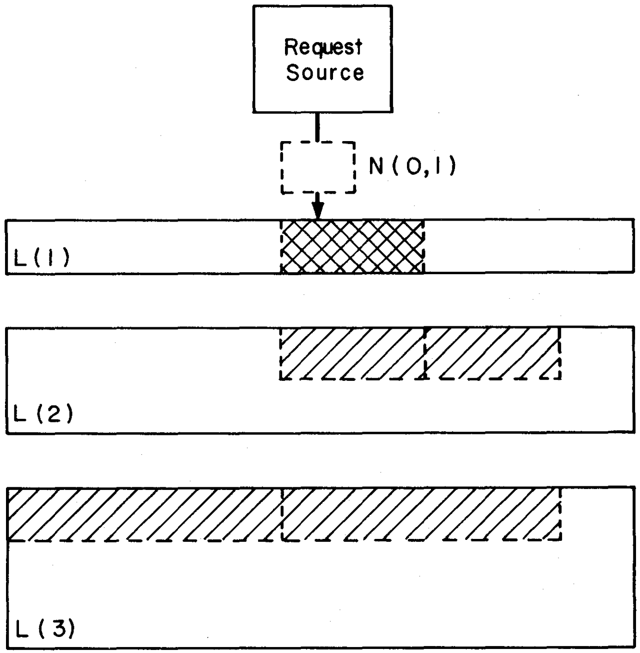


Figure 6—Example of the two-level store-behind process (a) A processor stores into $L(1)$, corresponding page is marked "changed"

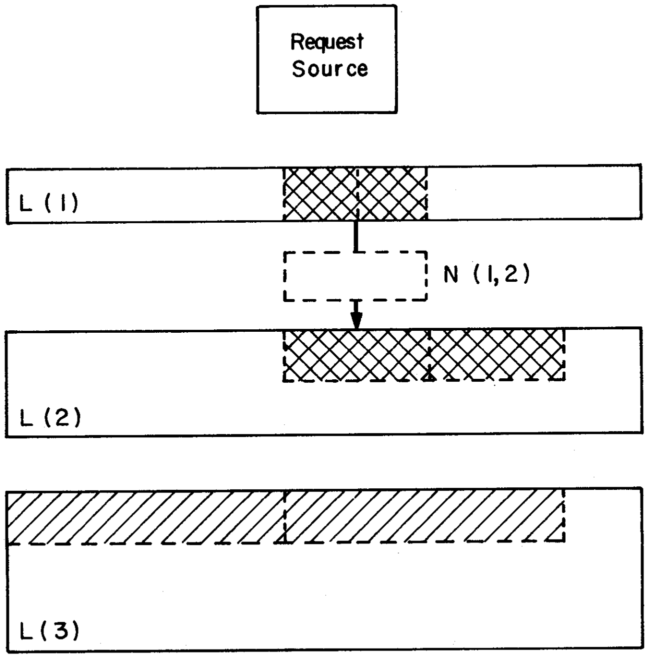


Figure 6(b)—At a later time, the corresponding page in $L(2)$ is updated and marked "changed"

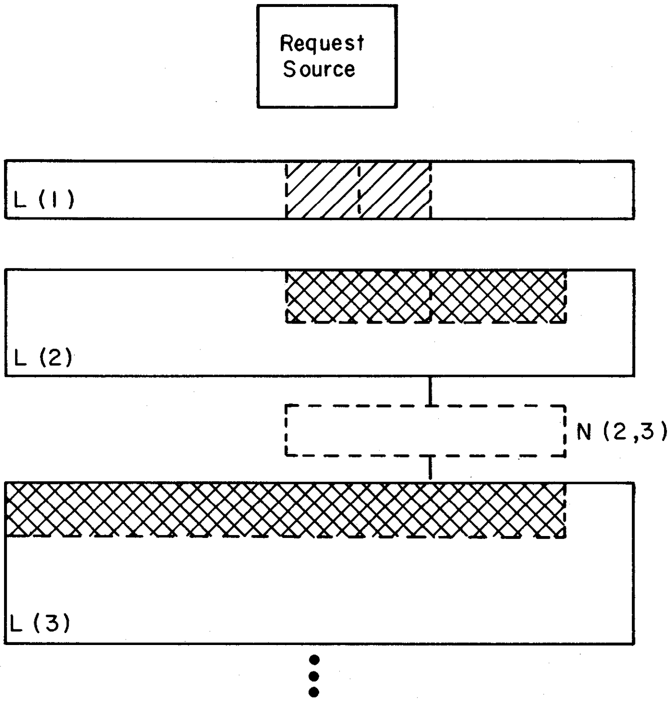


Figure 6(c)—At a later time, the corresponding page in $L(3)$ is updated and marked "changed." Since copies of the changed information exists in both $L(2)$ and $L(3)$, the "changed" indicator can be reset in $L(1)$ and that page may be replaced if necessary

example of this technique can be found in the IBM 3850 Mass Storage System.[1,10]

Decentralized control

In order to develop information systems capable of managing extremely large memories and processing a tremendous number of requests, a design based upon decentralized control is essential. It is unlikely that a single processor would be capable of maintaining a centralized control over the large number of high-speed asynchronous operations needed. Furthermore, a centralized control could be seriously impared if there were a reliability failure. The hierarchical decomposition theory represents a straightforward basis for a decentralized control design.

FURTHER RESEARCH

There are many areas of further research under investigation, such as:

Optimal function and physical decomposition. It is necessary to define a measure of performance and be able to prove that a particular decomposition is optimal.

Equivalence between functions. All information system functionalities must be mapped onto a standard set of functions. It is necessary to prove that the decomposed functionality is equivalent to the desired functionality.

Performance of physical implementation. There are various possible physical implementations of the optimal function and physical decomposition. It is necessary to provide measures for the performance of each such implementation and determine optimality.

Reliability. Although examples of reliability techniques have been described in this paper, a detailed reliability plan that encompasses all eventualities must be found.

Interlocks. Various interlock mechanisms must be used in an information system to coordinate various independent update operations. It is necessary to develop interlock techniques that lend themselves to a highly decentralized implementation without adversely affecting performance or reliability.

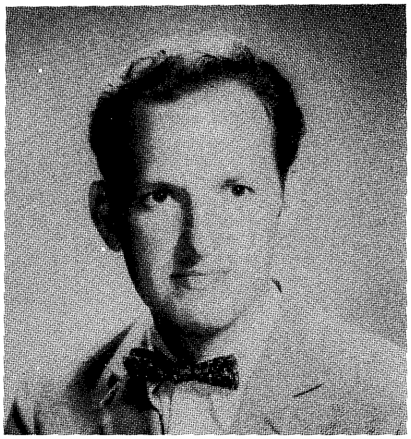
CONCLUDING COMMENTS

By using hierarchical physical decomposition (as depicted in Figure 3) to provide the virtual storage needed for the hierarchical function decomposition (as shown in Figure 1), a complete hierarchical decomposition of a large information management system can be attained. Furthermore, using the techniques illustrated in Figures 4 and 5, the hierarchical decomposition can be implemented by means of a microprocessor complex (i.e., the INFOPLEX).

Although such an INFOPLEX has not yet been completely implemented, portions are under development and investigation.[20] The overall theory of hierarchical decomposition remains an important research area.

REFERENCES

1. Ahearn, G. R., Y. Dishon, and R. N. Snively, "Design Innovations of the IBM 3830 and 2835 Storage Control Units," *IBM Journal of Research and Development,* Vol. 16, No. 1, pp. 11-18, January 1972.
2. Bensoussan, A., C. T. Clingen and R. C. Daley, "The Multics Virtual Memory," *Second ACM Symposium on Operating Systems Principles,* Princeton University, pp. 30-42, October 1969.
3. Chu, W. W. and H. Opderbeck, "Performance of Replacement Algorithms with Different Page Sizes," *Computer,* Vol. 7, No. 11, pp. 14-21, November 1974.
4. Considine, J. P. and A. H. Weis, "Establishment and Maintenance of a Storage Hierarchy for an On-line Data Base Under TSS/360," *FJCC,* Vol. 35, pp. 433-440, 1969.
5. Conti, C. J., "Concepts for Buffer Storage," *IEEE Computer Group News,* pp. 6-13, March 1969.
6. Denning, P. J., "Virtual Memory," *ACM Computing Surveys,* Vol. 2, No. 3, pp. 153-190, September 1970.
7. Donovan, J. J. and H. Jacoby, "A Hierarchical Approach to Information System Design," *MIT Sloan School of Management Report CISR-5 and WP 762-75,* January 1975.
8. Greenberg, Bernard S. and Steven H. Webber, "Multics Multilevel Paging Hierarchy," *IEEE INTERCON,* 1975.
9. Hatfield, D. J., "Experiments on Page Size, Program Access Patterns, and Virtual Memory Performance," *IBM Journal of Research and Development,* Vol. 16, No. 1, pp. 58-66, January 1972.
10. Johnson, Clay, "IBM 3850—Mass Storage System," *IEEE INTERCON,* 1975.
11. Madnick, S. E., "Design Strategies for File Systems," *MIT Project MAC Report TR-78,* October 1970.
12. Madnick, S. E., "Storage Hierarchy Systems," *MIT Project MAC Report TR-105,* Cambridge, Massachusetts, 1973.
13. Madnick, S. E. and J. J. Donovan, *Operating Systems,* McGraw-Hill, New York, 1974.
14. Madnick, S. E., "Design of a General Hierarchical Storage System," *IEEE INTERCON,* 1975.
15. Martin, R. R. and H. D. Frankel, "Electronic Disks in the 1980's," *Computer,* Vol. 8, No. 2, pp. 24-30, February 1975.
16. Mattson, R., et al., "Evaluation Techniques for Storage Hierarchies," *IBM Systems Journal,* Vol. 9, No. 2, pp. 78-117, 1970.
17. Meade, R. M., "On Memory System Design," *FJCC,* Vol. 37, pp. 33-44, 1970.
18. Ohnigian, Suran, "Random File Processing in a Storage Hierarchy," *IEEE INTERCON,* 1975.
19. Ornstein, S. M., et al., "PLURIBUS—A Reliable Multiprocessor," *NCC,* 1975.
20. Toong, H. D., "Microprocessor-based Multiprocessor Ring Structured Network," *NCC,* 1975.
21. Wensley, J. H., "The Impact of Electronic Disks on System Architecture," *Computer,* Vol. 8, No. 2, pp. 24-30, February 1975.
22. Withington, F. G., "Beyond 1984: A Technology Forecast," *Datamation* Vol. 21, No. 1, pp. 54-73, January 1975.



Area Director:
Donald C. Lincicome
Control Data Corporation
Sunnyvale, California

Interactive graphics

The two computer graphic sessions are to a high degree End User oriented.

The kinds of people who will find the "Data Base" session of interest are Application Analysts and End Users who with an Applications Analyst's interest is seeking new methods of data representation of physical models where the volume of data can grow quite large while access must remain rapid even though the access trail cannot be predicted or predetermined. The kinds of people interested in the "Economics of C.G." session are End Users and managers who must understand the areas of costs and productivity improvement measures that result from Operational Computer Graphics, and who want to learn from other Users experiences methods of economic justification and measures to ensure economic success of the necessary capital investment.

A polyhedron representation for computer vision

by BRUCE G. BAUMGART
Stanford University
Stanford, California

USE OF POLYHEDRA IN COMPUTER VISION

My approach to computer vision is best characterized as inverse computer graphics. In computer graphics, the world is represented in sufficient detail so that the image forming process can be numerically simulated to generate synthetic television images; in the inverse, perceived television pictures (from a real TV camera) are analyzed to compute detailed geometric models.

For example, the polyhedron in Figure 1 was computed from views of a plastic horse on a turntable by intersecting silhouette cones. As such, silhouette cone intersection is a purely descriptive vision technique; it is a form of wide angle stereo reconstruction. Like in the joke about carving a statue by cutting away everything that does not look like the subject, the approximate shape of the horse is hewed out of 3-D space by cutting away everything that falls outside of the silhouettes. In the example, the model was made from three silhouettes of the horse facing to the left which may be compared with two views of the horse facing to the right. One of the views is a real video image and the other is a display of the result showing how the process automatically constructed a backside for the horse consistent with the given silhouettes.

The present implementation requires a favorably arranged viewing environment (white objects on dark backgrounds or vice versa); application to more natural situations will be possible when a bulk correlation processor (an SPS-41) becomes available for extracting silhouettes by stereo depth discontinuities. Furthermore, the restriction to turntable rotation is for the sake of easy camera solving; this restriction will be lifted by providing stronger feature tracking for camera calibration. The silhouette cone intersection method can construct concave objects and even objects with holes in them; what are missed are concavities with a full rim, that is points on the surface of the object whose tangent plane cuts the surface in a loop that encloses the point. The idea arose out of an original intention to do "blob" oriented visual model acquisition, however a 2-D blob came to be represented by a silhouette polygon and a 3-D blob consequently came to be represented by a polyhedron.

Once acquired, a 3-D model can be used to anticipate the appearance of an object in a scene, making feasible a quantitative form of visual feedback. In Figure 2 for example, the approximate video appearance of the machine parts schematically depicted (top) can be computed and analyzed for edges (middle) and compared with an edge analysis of an actual video image of the parts (bottom). By comparing the predicted image with a perceived image, the correspondence between features of the internal model and features of the external reality can be established and a corrected location of the parts and the camera can be measured. Visually acquired 3-D geometric models can be of use to other robotic processes such as manipulation, navigation or recognition.

Unfortunately, these two approaches to computer vision (descriptive vision and verification vision) are only as strong as the state of the art in 3-D computer graphics. Consequently, my recent vision work has been largely concerned with the representation and manipulation of 3-D objects; objects which are solid, opaque and rigid. Although there are several significantly different geometric modeling ideas: arrays, 3-D density functions, 2-D parametric functions, volume elements, cross sectional elements, skeletons, manifolds and polyhedra; I have concentrated on polyhedra because they are simple enough to readily handle in a computer and complex enough to represent an arbitrary opaque surface. The rest of this paper is devoted to presenting a particular polyhedron representation for which convenient sets of manipulation routines have been developed.

INTRODUCTION TO THE WINGED EDGE

The Winged Edge polyhedron representation is implemented as a data structure composed of small blocks of words containing pointers and data in the fashion usual to graphics and simulation. An introduction to such data structures can be found in Chapter 2 of Knuth's Art of Computer Programming.[1] Quickly reviewing Knuth's terminology, a node is a group of consecutive words of memory, a field is a named portion of a node and a link is the machine address of a node. The notation for referring to a field of a node consists simply of the field name followed by a link expression enclosed in parentheses. For example, the two faces of an edge node whose link is stored in the variable named "edge", are found in the fields named NFACE and PFACE, and are referred to as

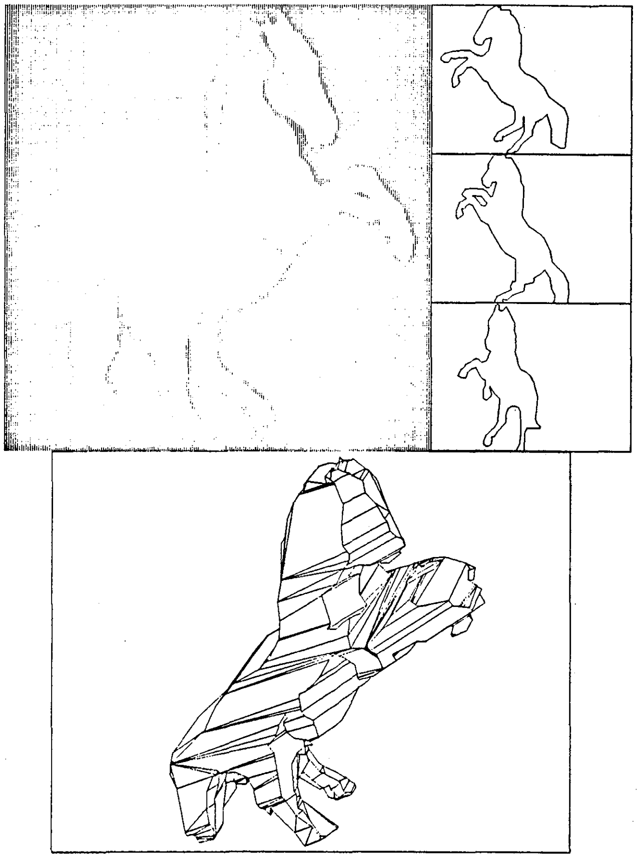


Figure 1—An example of silhouette cone intersection

NFACE(edge) and PFACE(edge). Although my latest language of implementation is PDP-10 machine code, examples will be given in a fictional programming language which combines ALGOL with Knuth's node/link notation.

A polyhedron is made up of four kinds of nodes: bodies, faces, edges and vertices. The body node is the head of three rings: a ring of faces, a ring of edges and a ring of vertices. In this context, a ring is a doubly linked circular list with a head node. Each face and each vertex points directly at only one of the edges on its perimeter. Each edge points to its two faces and its two vertices. Completing the topology, each edge node contains a link to each of its four immediate neighboring edges clockwise and counterclockwise about its face perimeters as seen from the exterior side of the surface of the polyhedron. These last four links are the wings of the edge, which provide the basis for efficient face perimeter and vertex perimeter accessing. Finally, the links of the edge nodes can be consistently oriented with respect to the surface of the polyhedron so that the surface always has two sides: the inside and the outside.

Observe that there are twenty-two link fields in the basic representation: bodies contain six links, faces three links, vertices three links and edges ten links. If we allow a link name such as PED to serve different roles depending on whether it applies to a body, face, edge or vertex; then the minimum number of different link field names that need to be coined is ten. The data structures and the link fields comprising the structures are listed in Figures 3 and 4. The ten link names include: NFACE and PFACE for two fields that contain face links in edges and the face ring, NED and PED for two fields that contain edge links, NVT and PVT for two fields that contain vertex links, and NCW, PCW, NCCW and PCCW for the four fields that contain edge links and are called the wings.

By constraining the arrangement of links in an edge node both the surface orientation (interior and exterior) and a linear orientation of the edge as a directed vector can be encoded. Figure 3 diagrams the arrangement of the links comprising the topology of an edge of a polyhedron

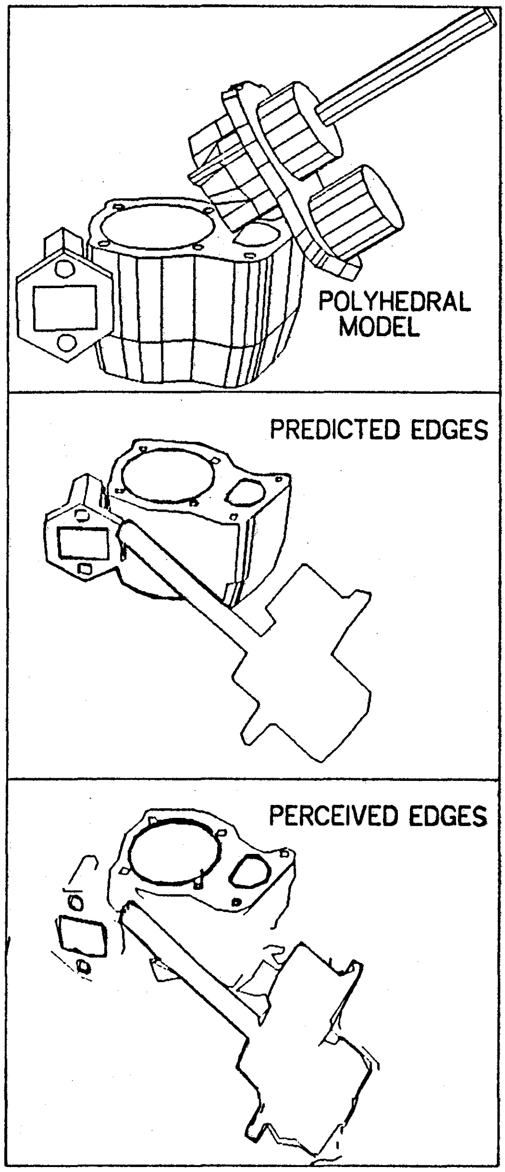
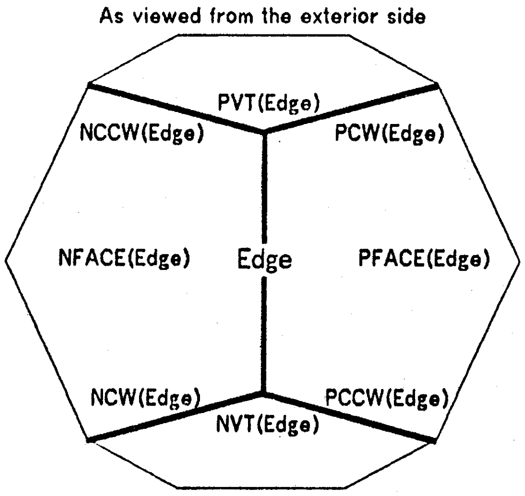


Figure 2—An example of verification vision

as viewed from the exterior side of its surface. Although the vertices in the figure are shown with only three edges, vertices may have any number of edges; the other potential edges would not be directly linked to the middle edge of the figure and so were not shown.

To complete the representation, space is allocated to contain 3-D coordinates of each vertex in fields named XWC, YWC and ZWC; the initials "WC" stand for *World Coordinates*. For the sake of vision and display, three more words are allocated to hold the *Perspective Projected coordinates* of each vertex in fields named XPP, YPP and ZPP. Also a word of thirty six status bits is carried in every node: permanent status bits specify the type (body, face, edge, vertex, etc.) of every node, temporary bits provide space for operations such as hidden line elimination that require marking. Passing now from necessities to conveniences, faces carry exterior pointing normal vectors and several words of photometric surface characteristics. The face vectors are derived from surface topology and vertex loci, and so they are not basic geometric data as in some representations. Bodies carry a print name, as well as four link fields (DAD, SON, BRO, SIS) for implementing a parts tree data structure; and two link fields (CW and CCW) for a body ring of all the bodies in the world model. Node formats are given in Figure 4 for an implementation based on fixed sized (twelve word) nodes.



1. Face ring of a body:
   NFACE(body or face) & PFACE(body or face).
2. Edge ring of a body:
   NED(body or edge) & PED(body or edge).
3. Vertex ring of a body:
   NVT(body or vertex) & PVT(body or vertex).
4. First edge of a face or vertex:
   PED(vertex) or PED(face).
6. The two faces of an edge:
   NFACE(edge) and PFACE(edge).
7. The two vertices of an edge:
   NVT(edge) and PVT(edge).
8. The four wing edges of an Edge:
   NCW(Edge) edge of NFACE Clockwise from Edge.
   PCW(Edge) edge of PFACE Clockwise from Edge.
   NCCW(Edge) edge of NFACE CCW from Edge.
   PCCW(Edge) edge of PFACE CCW from Edge.

Figure 3 Winged edge topology

The Winged Edge Polyhedron Representation as just presented is complete. Edge nodes carry most of the topology, vertex nodes carry the geometry, face nodes carry the photometry and body nodes carry the nomenclature and parts tree structure. The point that remains to be demonstrated, is that the appropriate subroutines for creating, maintaining and exploiting edge orientation execute efficiently and provide good primitives for solving such geometric problems as hidden line elimination and polyhedral intersection.

SEQUENTIAL ACCESSING

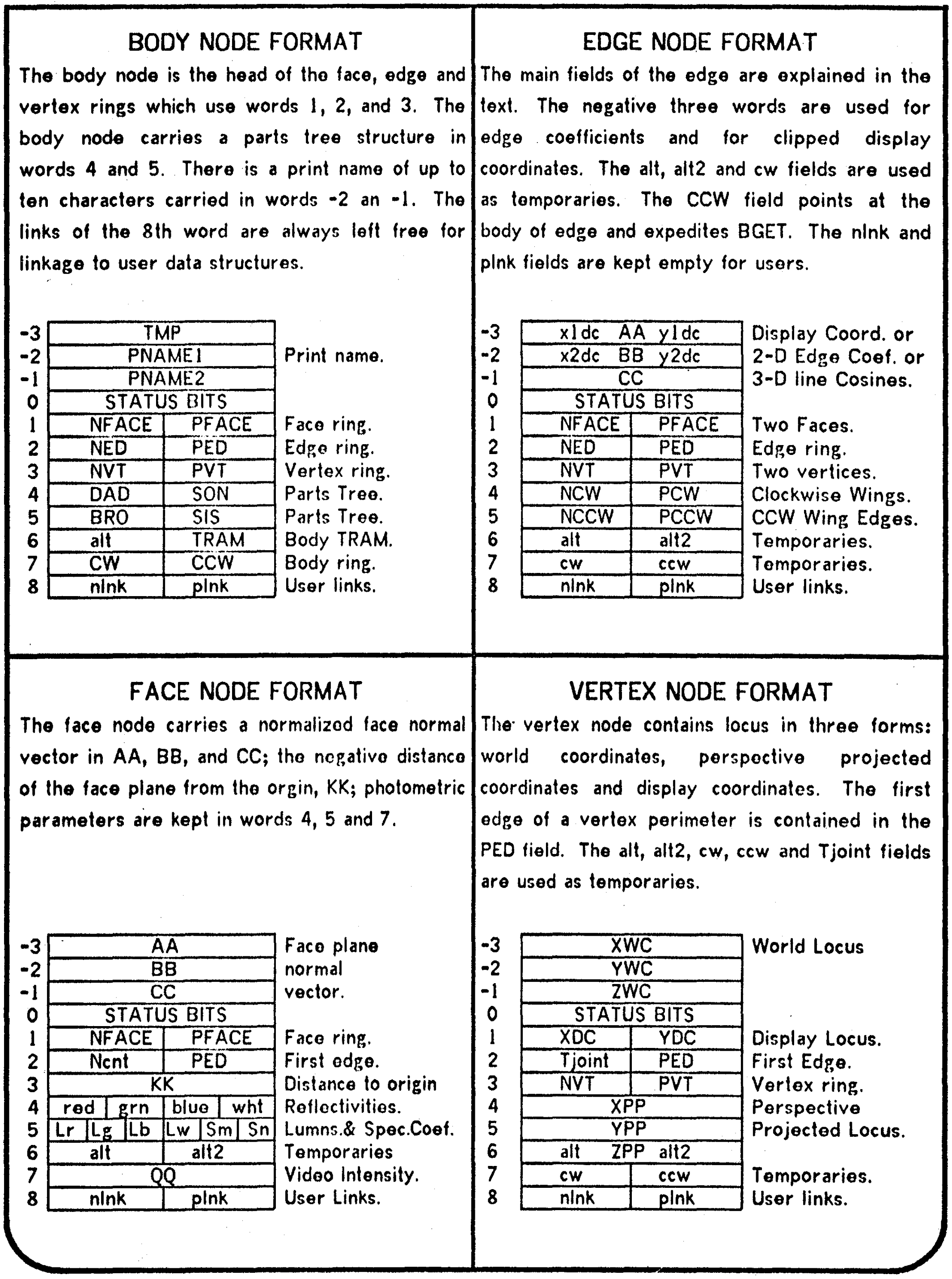
An immediate consequence of the ring structures is that the faces, edges and vertices of a body are sequentially accessible in the manner illustrated by the following lines of code:

```
COMMENT APPLY A FUNCTION TO ALL THE
FACES, EDGES AND VERTICES OF A BODY;
PROCEDURE APPLY (PROCEDURE FN;
INTEGER B);
BEGIN
INTEGER F,E,V;
  F←B; WHILE B≠(F←PFACE(F)) DO FN(F);
    COMMENT APPLY FUNCTION TO FACES OF
    A BODY;
  E←B; WHILE B≠(E←PED(E)) DO FN(E);
    COMMENT APPLY FUNCTION TO EDGES OF
    A BODY;
  V←B; WHILE B≠(V←PVT(V)) DO FN(V);
    COMMENT APPLY FUNCTION TO VERTICES
      OF A BODY;
END;
```

The rings could of course have been traversed in the other direction by invoking NVT, NED and NFACE in place of PVT, PED and PFACE. The reason for doubly linked lists (i.e., rings) is rapid deletion. Finally, observe that the face and vertex rings could be eliminated at the cost of having a more complicated face/vertex sequential accessing method requiring a visitation marking bit in the status word of face and vertex nodes.

PERIMETER ACCESSING

The perimeter of a face is an ordered list of edges and vertices, the perimeter of a vertex is an ordered list of



BODY NODE FORMAT

The body node is the head of the face, edge and vertex rings which use words 1, 2, and 3. The body node carries a parts tree structure in words 4 and 5. There is a print name of up to ten characters carried in words -2 an -1. The links of the 8th word are always left free for linkage to user data structures.

-3	TMP		
-2	PNAME1		Print name.
-1	PNAME2		
0	STATUS BITS		
1	NFACE	PFACE	Face ring.
2	NED	PED	Edge ring.
3	NVT	PVT	Vertex ring.
4	DAD	SON	Parts Tree.
5	BRO	SIS	Parts Tree.
6	alt	TRAM	Body TRAM.
7	CW	CCW	Body ring.
8	nlnk	plnk	User links.

EDGE NODE FORMAT

The main fields of the edge are explained in the text. The negative three words are used for edge coefficients and for clipped display coordinates. The alt, alt2 and cw fields are used as temporaries. The CCW field points at the body of edge and expedites BGET. The nlnk and plnk fields are kept empty for users.

-3	x1dc AA y1dc		Display Coord. or
-2	x2dc BB y2dc		2-D Edge Coef. or
-1	CC		3-D line Cosines.
0	STATUS BITS		
1	NFACE	PFACE	Two Faces.
2	NED	PED	Edge ring.
3	NVT	PVT	Two vertices.
4	NCW	PCW	Clockwise Wings.
5	NCCW	PCCW	CCW Wing Edges.
6	alt	alt2	Temporaries.
7	cw	ccw	Temporaries.
8	nlnk	plnk	User links.

FACE NODE FORMAT

The face node carries a normalized face normal vector in AA, BB, and CC; the negative distance of the face plane from the orgin, KK; photometric parameters are kept in words 4, 5 and 7.

-3	AA		Face plane
-2	BB		normal
-1	CC		vector.
0	STATUS BITS		
1	NFACE	PFACE	Face ring.
2	Ncnt	PED	First edge.
3	KK		Distance to origin
4	red | grn	blue | wht	Reflectivities.
5	Lr | Lg | Lb	Lw | Sm | Sn	Lumns.& Spec.Coef.
6	alt	alt2	Temporaries
7	QQ		Video Intensity.
8	nlnk	plnk	User Links.

VERTEX NODE FORMAT

The vertex node contains locus in three forms: world coordinates, perspective projected coordinates and display coordinates. The first edge of a vertex perimeter is contained in the PED field. The alt, alt2, cw, ccw and Tjoint fields are used as temporaries.

-3	XWC		World Locus
-2	YWC		
-1	ZWC		
0	STATUS BITS		
1	XDC	YDC	Display Locus.
2	Tjoint	PED	First Edge.
3	NVT	PVT	Vertex ring.
4	XPP		Perspective
5	YPP		Projected Locus.
6	alt ZPP alt2		
7	cw	ccw	Temporaries.
8	nlnk	plnk	User links.

Figure 4—Example of winged edge node formats

edges and faces, and the perimeter of an edge is an ordered list consisting of exactly two faces and two vertices. The perimeter definitions are caricatured in Figure 5. One virtue of the winged edge representation is that both vertex and face perimeters can be traversed in either direction (clockwise or counterclockwise) while being dynamically maintained in "*one ring*".

Given one edge of a face (or vertex) perimeter, the next edge clockwise (or counterclockwise) from the given edge about the particular face (or vertex) can be retrieved from the data structure with the assistance of two subroutines called ECW and ECCW. The idea of the edge clocking routines is to match the given face (or vertex) with one of the faces (or vertices) of the given edge and to then return the appropriate wing. A possible coding of ECCW and ECW might be as follows:

```
COMMENT FETCH EDGE CCW FROM E ABOUT
FV;
INTEGER PROCEDURE ECCW (INTEGER E,FV);
BEGIN "ECCW"
  IF PFACE(E)=FV THEN RETURN(PCCW(E));
  IF NFACE(E)=FV THEN RETURN(NCCW(E));
  IF PVT(E)=FV    THEN RETURN(PCW(E));
  IF NVT(E)=FV    THEN RETURN(NCW(E));
  FATAL;
END "ECCW";
```

```
COMMENT FETCH EDGE CLOCKWISE FROM E
ABOUT FV;
INTEGER PROCEDURE ECW (INTEGER E,FV);
BEGIN "ECW"
  IF PFACE(E)=FV THEN RETURN(PCW(E));
  IF NFACE(E)=FV THEN RETURN(NCW(E));
  IF PVT(E)=FV    THEN RETURN(NCCW(E));
  IF NVT(E)=FV    THEN RETURN(PCCW(E));
  FATAL;
END "ECW";
```

The first edge of a face or vertex is (of course) immediately available from the PED field of the face or vertex. For example, the two procedures below can be used to visit all the edges of a face or all the edges of a vertex, respectively.

```
COMMENT APPLY FUNCTION TO EDGES OF A
FACE;
PROCEDURE APPLY (PROCEDURE FN;
INTEGER F);
BEGIN
  INTEGER E,E0;
  E←E0←PED(F);
  DO FN(E) UNTIL EØ=(E←ECCW(E,F));
END;
```

```
COMMENT APPLY FUNCTION TO EDGES OF A
VERTEX;
PROCEDURE APPLY (PROCEDURE FN;
INTEGER V);
BEGIN
  INTEGER E,E0
  E←E0←PED(V);
  DO FN(E) UNTIL E0=(E←ECCW(E,V));
END;
```

Using the same idea as in the edge clocking routines, a face or vertex can be retrieved relative to a given edge and a given face or vertex. These routines include; FCW and FCCW which return the face clockwise or counterclockwise from a gvien edge with respect to a given vertex; VCW and VCCW which return the vertex clockwise or counterclockwise from a given edge with respect to a given face; and OTHER which returns the face or vertex of the given edge opposite the given face or vertex. Together the seven routines: ECW, ECCW, VCW, VCCW, FCW, FCCW and OTHER exhaust the possible oriented retrievals from an edge node; they also alleviate the need to ever explicitly reference a wing field when traveling the surface or a polyhedron. With node type checking the primitives can be made stronger, for example ECCW(vertex,face) is implemented to return the edge counterclockwise from the given vertex about the given face. With node type checking and signed arguments the seven perimeter accessing routines could even be replaced by a single routine perhaps named PERIMETER_FETCH or PGET. On the other hand, I favor having the proliferation of accessing names for the sake of documenting the clocking direction and the types of nodes involved.

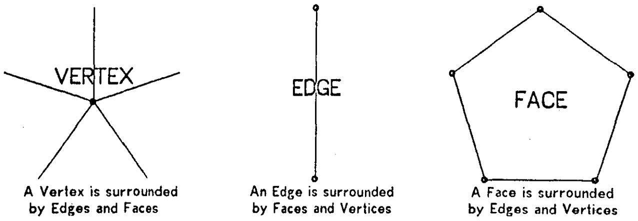

VERTEX
EDGE
FACE
A Vertex is surrounded by Edges and Faces
An Edge is surrounded by Faces and Vertices
A Face is surrounded by Edges and Vertices

Figure 5—Three kinds of perimeters

Two remaining accessing routines, of minor importance, are BGET(entity) and LINKED(entity,entity). BGET of a face, edge or vertex merely cycles the appropriate ring to retrieve the body of the given entity. The LINKED routine determines whether its two arguments (faces, edges or vertices) are adjacent; there are six LINKED cases: (i) Face-Face, returns a common edge or FALSE; (ii) Face-Edge, returns Boolean value F=PFACE(E) or F=NFACE(E); (iii) Edge-Edge, returns a common vertex or false; (v) Edge-Vertex, returns Boolean value V=PVT(E) or V=NVT(E); (vi) Vertex-Vertex, returns common edge or FALSE. (As in LISP, zero is false and nonzero is true).

BASIC POLYHEDRON SYNTHESIS

LOWEST LEVEL WINGED EDGE ROUTINES.
Node Makers: MKNODE, MKB, MKF, MKE, MKV, MKTRAM.
Node Killers: KLNODE, KLB, KLF, KLE, KLV.
Wing Mungers: WING, INVERT, EVERT.
Surface Fetchers: ECW, ECCW, OTHER, VCW, VCCW, FCW, FCCW, LINKED.
Parts Tree Routines: BDET, BATT, BGET.

There are sixteen routines for node creation and link manipulation which when combined with the nine accessing routines of the previous section form the nucleus of a polyhedron modeling system. These routines are very low level in that the final applications user of winged polyhedra will never explicitly need to make a node or mung a link. The word *mung* (meaning to modify an existing structure by altering links in place) is LISP slang that deserves to be promoted into the technical jargon; traditionally, a mung routine is one which makes applications of the LISP primitives RPLACA and RPLACD. The twenty five routines listed above are the bedrock for the Euler primitives, which are an elegant set of subroutines for altering polyhedra while always maintaining the Euler relation: $F-E+V=2*B-2*H$ between the numbers of bodies, faces, edges, vertices and handles. Examples of Euler primitives are given in another paper written for this conference[2] as well as Section 3 of Reference 3 and so will not be elaborated here.

Node makers and killers

The MKNODE and KLNODE are the raw storage allocation routines which fetch or return a node from the available free storage. The MKB routine creates a body node with empty face, edge and vertex rings; the body is placed into the body ring of the world model. The MKF, MKE and MKV each take one argument and create a new face, edge or vertex node in the ring of the given entity: with type checking these three primitives could be consolidated. Finally the MKTRAM node creates a *tram node*, which consists of twelve real numbers that represent either a Euclidean transformation or a Cartesian frame of reference depending on the context. As a Cartesian frame of reference the tram node is interpreted as a 3-D locus in world coordinates with a right handed triad of orthogonal unit vectors; as a Euclidean transformation the tram node is interpreted as a translation vector followed by a rotation matrix. Tram nodes are further explained in Reference 3. The corresponding kill routines KLB, KLF, KLE and KLV remove the entity from its respective ring and return its node to free storage.

Wing mungers

The WING(edge1,edge2) routine finds that face and vertex the arguments edge1 and edge2 have in common and stores the wing pointers between edge1 and edge2 accordingly; the exact link manipulations are illustrated in the example coding of the WING procedure immediately following this paragraph. Recalling that edges are directed vectors, the INVERT(E) routine flips the direction of an edge by swapping the contents of the appropriate fields as follows: PFACE(E)↔NFACE(E); PVT(E)↔NVT(E); NCW(E)↔NCCW(E) and PCW(E)↔PCCW(E). Finally, the EVERT(B) routine turns a body inside out, by performing the following link swaps on all the edges of the given body: PFACE(E)↔NFACE(E); NCW(E)↔PCCW(E); and NCCW(E)↔PCW(E).

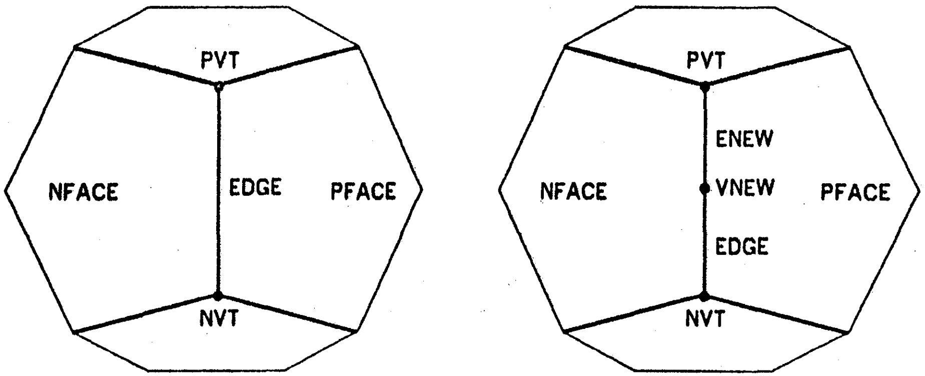
```
PROCEDURE WING(INTEGER E1,E2);
BEGIN
  IF PVT(E1)=PVT(E2)∧PFACE(E1)=
    NFACE(E2) THEN BEGIN PCW(E1)←E2;
    NCCW(E2)←E1;END;
  IF PVT(E1)=PVT(E2)∧NFACE(E1)=
    PFACE(E2) THEN BEGIN NCCW(E1)←E2;
    PCW(E2)←E1;END;
  IF PVT(E1)=NVT(E2)∧PFACE(E1)=
    PFACE(E2) THEN BEGIN PCW(E1)←E2;
    PCCW(E2)←E1;END;
  IF PVT(E1)=NVT(E2)∧NFACE(E1)=
    NFACE(E2) THEN BEGIN NCCW(E1)←E2;
    NCW(E2)←E1;END;
  IF NVT(E1)=PVT(E2)∧PFACE(E1)=
    PFACE(E2) THEN BEGIN PCCW(E1)←E2;
    PCW(E2)←E1;END;
  IF NVT(E1)=PVT(E2)∧NFACE(E1)=
    NFACE(E2) THEN BEGIN NCW(E1)←E2;
    NCCW(E2)←E1;END;
  IF NVT(E1)=NVT(E2)∧PFACE(E1)=
    NFACE(E2) THEN BEGIN PCCW(E1)←E2;
    NCW(E2)←E1;END;
  IF NVT(E1)=NVT(E2)∧NFACE(E1)=
    PFACE(E2) THEN BEGIN NCW(E1)←E2;
    PCCW(E2)←E1;END;
END;
```

Part tree routines

Body nodes can be grouped into a tree structure of parts. The parts tree consumes four link positions (DAD, SON, BRO, SIS) and is maintained in body nodes by the following primitives: BDET(body) detaches a body node from the parts tree, BATT(body1,body2) attaches body1 to the ring of children belonging to body2, and BGET(entity) returns the body node at the head of the given face, edge or vertex ring. The SON field of a body may contain a pointer to a headless ring of subpart bodies, the ring of subparts is maintained in the BRO (brother) and SIS (sister) fields, and each subpart contains a pointer back to its parent in its DAD field. At present, the notion of a body is coincident with the notion of a connected polyhedron; however by allowing several bodies to be associated with a single polyhedral surface, a flexible object such as an animal could be represented.

EDGE AND FACE SPLITTING

The most important property of the winged edge representation is that edges and faces can be split using subroutines that make only local alterations to the data structure; and the splits can easily be removed. The edge



BEFORE: VNEW ← ESPLIT(EDGE);
AFTER: EDGE ← KLEV(VNEW);

AFTER: VNEW ← ESPLIT(EDGE);
BEFORE: EDGE ← KLEV(VNEW);

```
INTEGER PROCEDURE ESPLIT (INTEGER EDGE);
BEGIN "ESPLIT"
        INTEGER VNEW,ENEW;
COMMENT CREATE A NEW EDGE AND VERTEX;
        VNEW ← MKV(PVT(EDGE));
        ENEW ← MKE(EDGE);
COMMENT CONNECT VERTICES & FACES TO EDGES;
        PVT(ENEW) ← PVT(EDGE);
        NVT(ENEW) ← VNEW;
        PVT(EDGE) ← VNEW;
        PFACE(ENEW) ← PFACE(EDGE);
        NFACE(ENEW) ← NFACE(EDGE);
COMMENT CONNECT EDGES TO VERTICES;
        IF PED(PVT(EDGE)=EDGE THEN
          PED(PVT(EDGE))←ENEW;
        PED(VNEW)←ENEW;
COMMENT LINK THE WINGS TOGETHER;
        NCW(ENEW) ← EDGE; PCCW(ENEW) ← EDGE;
        PCW(EDGE) ← ENEW; PCCW(EDGE) ← ENEW;
        WING(NCCW(EDGE),ENEW);
        WING(PCW(EDGE),ENEW);
        RETURN(VNEW);
END "ESPLIT";
```

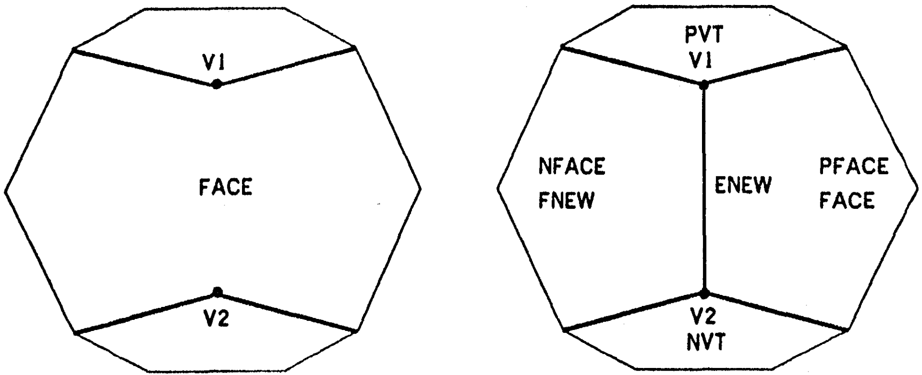
```
INTEGER PROCEDURE KLEV (INTEGER VNEW);
BEGIN "KLEV"
        INTEGER EDGE,ENEW,V,F,B;
        ENEW ← PED(VNEW);
        EDGE ← ECCW(ENEW,VNEW);
COMMENT ORIENT EDGES AS IN DIAGRAM;
        IF NVT(ENEW) ≠ VNEW THEN INVERT(ENEW);
        IF PVT(EDGE) ≠ VNEW THEN INVERT(EDGE);
COMMENT TIE E TO ITS NEW UPPER VERTEX AND WINGS;
        V ← PVT(EDGE) ← PVT(ENEW);
        WING(PCW(ENEW),EDGE);
        WING(NCCW(ENEW),EDGE);
COMMENT ELIMINATE OCCURRENCES OF ENEW IN F AND V;
        IF PED(V)=ENEW THEN PED(V) ← EDGE
        IF PED(PFACE(EDGE))=ENEW THEN
          PED(PFACE(EDGE))←EDGE;
        IF PED(NFACE(EDGE))=ENEW THEN
          PED(NFACE(EDGE))←EDGE;
COMMENT REMOVE NODES FROM RINGS AND RETURN EDGE;
        KLV(VNEW);
        KLE(ENEW);
        RETURN(EDGE);
END "KLEV";
```

The actual routines differ slightly from those given above in that they do argument type checking and data structure checking; nevertheless, a diagnostic trace of the implemented version reveals that the ESPLIT routine executes an average of 170 PDP-10 instructions and the KLEV routine executes an average of 200 instructions.

Figure 6—Make and kill edge-vertex

split routine, ESPLIT, makes a new edge and a new vertex and places them into the surface topology as shown in Figure 6; the kill edge-vertex routine, KLEV, undoes an ESPLIT. The face split routine, MKFE, creates a new edge and a new face and places them into the surface topology as shown in Figure 7; the kill face-edge routine, KLFE, undoes a MKFE.

The rest of this section concerns implementation, the use of the split and kill routines illustrate a pattern which applies to the coding of any operations on winged edge structures. In a typical situation, there are five steps: first, get the proper kinds of nodes into the body rings using the MKF, MKE, MKV primitives; second, position the vertices by setting their XWC, YWC, ZWC fields; third, connect each vertex and face to one of its edges by setting face/vertex PED fields; fourth, connect each edge to its two faces and its two vertices by setting the NFACE, PFACE, NTV, PVT fields of the edge; finally, set up the wing perimeter pointers by applying the WING primitive to the pairs of edges to be mated.



BEFORE: ENEW ← MKFE(V1,FACE,V2);
AFTER: FACE ← KLFE(ENEW);

AFTER: ENEW ← MKFE(V1,FACE,V2);
BEFORE: FACE ← KLFE(ENEW);

```
INTEGER PROCEDURE MKFE(INTEGER V1,FACE,V2);
BEGIN "MKFE"
        INTEGER V1,V2,FNEW,ENEW,E,E0,B,V;
COMMENT CREATE NEW FACE & EDGE;
        FNEW ← MKF(FACE); ENEW ← MKE(PED(FACE));
COMMENT LINK NEW EDGES TO ITS FACES & VERTICES;
        PED(F) ← PED(FNEW) ← ENEW;
        PFACE(ENEW) ← F; NFACE(ENEW) ← FNEW;
        PVT(ENEW) ← V1; NVT(ENEW) ← V2;
COMMENT GET THE WINGS OF THE NEW EDGE;
        E2 ← PED(V1);
        DO E2←ECW((E1←E2),V1) UNTIL FCW(E1,V1)=FACE;
        E4 ← PED(V1);
        DO E4←ECW((E3←E4),V2) UNTIL FCW(E3,V2)=FACE;
COMMENT SCAN CCW FROM V1 REPLACING F'S WITH FNEW;
        E ← E2;
        DO IF PFACE(E)=FACE THEN PFACE(E)←FNEW
         ELSE NFACE(E)←FNEW;
        UNTIL E4 = (E←ECCW(E,FNEW));
COMMENT LINK THE WINGS;
        WING(E1,ENEW); WING(E2,ENEW);
        WING(E3,ENEW); WING(E4,ENEW);
        RETURN(ENEW);
END;
```

```
INTEGER PROCEDURE KLFE (INTEGER ENEW);
BEGIN "KLFE"
        INTEGER FNEW,FACE,V1,V2,E,E1,E2,E3,E4;
COMMENT PICKUP ALL THE LINKS OF ENEW;
        FACE ← PFACE(ENEW); FNEW ← NFACE(ENEW);
        V1 ← PVT(ENEW);  V2 ← NVT(ENEW);
        E1 ← PCW(ENEW); E2 ← NCCW(ENEW);
        E3 ← NCW(ENEW); E4 ← PCCW(ENEW);
COMMENT GET ENEW LINKS OUT OF FACE, V1 AND V2;
        IF PED(V1) = ENEW THEN PED(V1) ← E1;
        IF PED(V2) = ENEW THEN PED(V2) ← E3;
        IF PED(FACE)=ENEW THEN PED(FACE)←E3;
COMMENT GET RID OF FNEW APPEARANCES;
        E ← E2;
        DO IF PFACE(E)=FNEW THEN PFACE(E)←FACE
         ELSE NFACE(E)←FACE;
        UNTIL E4 = (E←ECCW(E,FNEW));
COMMENT LINK WINGS TOGETHER ABOUT FACE;
        WING(E2,E1);WING(E4,E3);
        KLF(FNEW);KLE(ENEW);
        RETURN(FACE);
END;
```

Again, the actual routines differ from those given above in that they do argument type checking and data structure checking. The above two routines typically take about twice as long to execute as the previous pair; notice that the execution time is dependent on the length of face perimeters, which are mostly three or four edges long.

Figure 7—Make and kill face-edge

CONCLUSION

The technical point of this paper is that a polyhedral representation with a coherent and locally alterable topology can be constructed. The larger philosophical point is that computer vision perhaps can be realized by using computer graphics techniques to keep an internal mental simulation in sync with the changing appearance of the external physical reality.

REFERENCES

1. Knuth, Donald Ervin, *The Art of Computer Programming*, Addison-Wesley, Reading, Massachusetts, 1968.
2. Eastman, Lividini & Stoker, "A Database for Designing Large Physical Systems," *Proceedings of the National Computer Conference*, May 1975.
3. Baumgart, Bruce G., *Geometric Modeling for Computer Vision*, Stanford Artificial Intelligence Laboratory, Memo no. AIM-249, Stanford University, October 1974.

Aspects of modelling in computer aided geometric design*

by RICHARD F. RIESENFELD
University of Utah
Salt Lake City, Utah

INTRODUCTION

This paper identifies some problems of modelling free-form geometry in a computer, and discusses attributes of a good model and good parameters for a model. In particular, we are interested in representing arbitrary shape information, shapes that may not have special names and may not be well defined except at particular points (or curves) of interest occasionally referred to as "*hard points*" or "*hard constraints.*" There may remain considerable freedom in modelling the "*soft points*" or "*soft constraints.*" Paradigms of this problem include modelling an automobile fender, a shoe last, or a boat hull. This is an aspect of an area that Forrest calls *computational geometry.*[12]

PROPERTIES OF GEOMETRIC MODELS

Devising adequate and useful *mathematical models* for representing geometric *shape information* has been an area of lively research even before a numerically controlled drafting machine was first logically attached to a computer. The straight line segment AB is easily represented by its end points (A, B); the polygonal line is then $(A_1, A_2, \ldots, A_n)$, where A_i is a vertex. Additional structure leads to economies in representing *line drawings,* such as architectural plans. Treating the plan like a large graph, we list all of the vertices in one array and describe the *connectivity* (also called "topology") in another array, known in graph theory as the connection matrix. Obviously this approach is not restricted to planar graphs.

Since the graphic output from a computer is quantized, albeit rather finely in some cases, it may seem no loss to quantize the input. Thus a very straightforward approach to representing a *free-form curve* is to approximate it by a polygonal line whose sides have length close to the size of the output quantization level, namely a *raster unit* or one lowest quantum unit of output resolution. Chain encoding[14] sample data points from a curve affects a major data reduction by storing mostly relative (incremental) information instead of the entire absolute coordinate information. Even a computing environment with rich enough resources to support a massive "sample and store" approach to representing curves cannot overcome the semantic loss that such a method engenders. This model leads to copious volumes of coordinate data in which much of the original relational and other attribute information might be irretrievably lost. Answering a simple user interrogation regarding similarity or identity of two quantized curves becomes a formidable task with this model.

A *closed form* like $x^2+y^2=5.7$ that is, a finite expression that describes a *genetic point* on a curve is more compact to store and more useful for identifying attributes of a curve. For example, one could *symbolically compare* the two closed forms to determine relationships between two modelled curves. A representation in the form of a list of points may not be necessary until the last step that generates a *display file* for driving a graphical output device like a plotter or a CRT.

Following the above advice is not a simple matter if we observe that a sketched curve almost never has an exact closed form. First we must adopt a satisfactory model for representing arbitrary curves in a computer. According to an internal view of the computer, we seek a mathematical model that allows us to represent arbitrary geometric curves as a short *list of parameters,* a form amenable to the linear stores found in most general purpose computers. In an abstract sense, this is a quantization inasmuch as we are representing information about infinitely many geometric points on a curve by finitely many parameters. Usually these parameters are coefficients (say, of a polynomial) that specify a "point" function in a *finite dimensional subspace* of the *infinite dimensional* space of continuous ($C^{[0]}$) curves, the *discretization* being the *integral dimension* of the subspace, or equivalently, the integral length of the corresponding parameter list. So the finiteness of a polynomial representation manifests itself by its restricted capacity for approximating complex, randomly meandering curves.

While we recognize that some inaccuracies will result from modelling (approximating) an infinite dimensional *object space,* we would normally insist on a scheme that affords us the power to come arbitrarily close, in the user's eye, to the object that we are trying to represent. In topological terms, we need a *model space that is dense in the object space* analogous to the way that the rational num-

* This work has been supported in part by the National Science Foundation (DCR74-13017) and the Advanced Research Projects Agency (DAHC15-73-C-0363).

bers are dense in the real numbers—every neighborhood (interval) of a real number contains a rational number.

In the case of a polynomial curve model, Weierstrass's Theorem assures us that we can come arbitrarily close in *maximum deviation,* or absolute distance, to any continuous curve in our object space. However, this does not guarantee that we can have a *smooth approximation,* only a close one. For smooth approximation we must base our model on a scheme like Bernstein or Bézier approximation,[4,15] where *shape* is approximated, as well as *position.*

INTERACTIVELY SPECIFYING GEOMETRIC MODELS

The discussion in this section has a rather passive existential tone, deferring the actual problem of dynamically constructing a particular model. While the *power of a model,* the class of objects that it can reasonably represent, may be sufficiently general, the model may be intractable to use because of difficulties that arise when one attempts to find the appropriate parameters for modelling an object of interest, for *specifying a model.*

Not only do we require that a model be capable of representing a wide class of objects of interest, we emphatically require that the model offer a *convergent procedure* which enables a user to specify a satisfactory parameter list. By analogy, the power of a programming language is a separate property from the programming diagnostics that guide the user along a convergent iteration of programs until a working program is achieved. The theoretical power of a programming language is not affected by the efficacy of the associated error messages. While this distinction is patently obvious in the programming language setting, the absence of useful and meaningful feedback from many systems that support geometric models brings this writer to devote some space to the idea.

In order to provide the user with a convergent procedure for specifying a model, that is, for making a *geometric statement,* we require a system whose controls correspond in some basic and predictable fashion to the geometric notions that a designer wishes to express. Is the *geometric language* provided suitable for communicating a geometric statement in a direct, efficient, and undistracting manner? Moreover, the control response should be smooth and stable; small perturbations of the input controls should result in small perturbations to the model, for otherwise convergence is left to serendipity alone. More abstractly, the input controls should be handles of *continuous operators,* like scaling or rotating, and *discrepancy information* offered to the user whenever possible. Failing in this, the system will be draped with a pall of frustration and rejection.

Ambiguity frequently occurs in the process of defining a geometric model, when the extent of the region of an intended modification is not explicit. Do we wish to raise a small bump in a restricted neighborhood of a point, a *local change,* or do we mean to raise the elevation of the entire object, a *global change*? Providing the facility in a *graphical editor* for conveniently specifying the affected area during an alteration is an important and subtle problem in graphical communications. Recently there has been considerable effort in developing various geometric models for curves and surfaces that allow for strictly local changes.

By employing these models it is possible to insure that, outside of a specified neighborhood, the original model is absolutely unchanged. The absence of this property from many earlier models is considered a serious deficiency in them. In mathematical terms, allowing local alterations means that the models are *piecewise* defined.

One of the most inhibiting restrictions in defining and viewing 3-dimensional geometry models is that most graphical input and output is through 2-D media like CRT's, plotters, and tablets. Thus the subtle communication problems just mentioned are severely magnified by having to express a statement about 3D objects in terms of multiple 2D projections. Similarly the visual impact and accuracy of a 2D rendering are often inadequate for supporting a graphical dialogue involving explorations in 3D.

Typically the scope of interest and degree of tolerances range widely during an interactive design session when a user tries to specify a model. By definition, design implies a mixture of *design constraints* and *design parameters,* the designer's freedoms. Designing a mechanical part like a casing may involve meeting certain boundary requirements in order to make the casing fit properly. On the other hand, broad freedom may be allowed in the shape of the casing apart from the boundary specifications. In summary the option for specifying parameters very precisely is essential, but the tedium of specifying everything in precise, minute details when it is not a special concern is a dissuasive process. One effective method for coping with what seems to be a dilemma between exacting design in the small, and casual design in the large, is a strategy for assigning reasonable *default parameters* in the model when the designer does not specify overriding particular values. Whenever possible the model should automatically be augmented with sensible default parameters that produce a nondegenerate model. The criteria for establishing such parameters depend very heavily on the specific nature of the design work, but in a given application it is often apparent what these parameters ought to be. In this way a designer only has to delineate initial information in the regions where he is primarily concerned about the shape. As he refines his design he can improve the default parameters to suit his special needs. Recently P. Dube[9] has obtained impressive results in this area of preliminary design.

Finally we note some of the repercussions that the previous discussion makes on the language that is used to express a graphical idea. Since sketching is a *natural idiom* for indicating shape information, the language must allow for a *pictorial statement.* The system has to distinguish between basic *essential hand motions* that define the basic character lines of an object, and *nonessential marks* that give depth, texture, and other qualities that contribute to the total visual impression on the observer. Of equal importance is the ability of the system to impose appropriate design constraints, to ignore *unintended impressions* that a designer does not wish to make. For instance, the random deviations (*noise*) from a true straight line should not be taken literally if a straight line is intended.

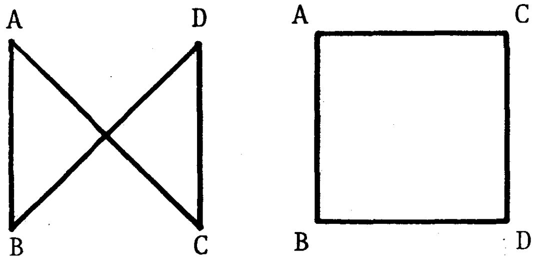
Work on developing graphical constraints in a system that supports some of the above notions dates back to Sketchpad.[24] In recent years major advances in recognizing and interpreting graphical gestures and subtleties were achieved by Negroponte's Machine Architecture Group at M.I.T.[18,19] and Baumgart[2,3] at Stanford's Artificial Intelligence Laboratory. In order to assist and interpret sketches, the graphics system must house and maintain a current intimate *model of the user* that includes the type of design work that the user is *currently* performing together with the user's own personal design idiosyncracies. The *context* of a motion is often the best clue toward interpreting its meaning. Research in this area joins computer graphics with artificial intelligence in a way that poses challenging problems for both areas.

Unless *appearance* is the sole concern, the geometric attributes of a model form one aspect of a more general *integrated model*[29] that contains additional information about structure, material, cataloging, and the like. For reasons beyond appearance, it is often necessary to specify very precise relationships, ones that are too fine for the eye to detect on a quantized output device. Such statements lend themselves to *character string input commands* because sketching information or pointing with a pen are too ambiguous in these applications.[30] So in addition to the special requirements that graphics imposes, we have to consider the remaining requirements that a particular nongraphical application brings on.

An appropriate keyboard language for issuing graphical commands is also necessary for forming higher level expressions for naming algebraic combinations of previously defined objects. A particular curve may only be defined as the intersection of two objects. Or a part may be simply defined as the value of a recursive algebraic expression involving additions and subtractions of other objects.[5,30] Work of this kind has been hindered by the massive *numerical and relational calculations* needed to evaluate such expressions.[29] But the imperative for admitting recursively defined and *hierarchically structured* objects in a graphics language is clear.

MODELLING AND RENDERING

In modelling a situation we strive to abstract the *semantic information* essential to the particular purpose at hand, while generally ignoring other structure that is nonessential to our needs. For example, we can satisfactorily represent the two graphs in Figure 1 with exactly the same connection array. Many situations may get mapped into a model with the identical parameter list, but we should be indifferent to the ambiguity if the model we chose is an appropriate one relative to our requirements. Figure 1 provides two representatives of the entire equivalence class of situations which have an identical connection array. Moreover, if the model is rich enough in its abstraction, we can define operations on the models that hold for corresponding operations in our actual experiences. If our model admits such operations and yields a suitable representative of the transaction, then we say that we have a *homomorphic model.*



$$\begin{array}{c|cccc|}
 & A & B & C & D \\
A & 0 & 1 & 1 & 0 \\
B & 1 & 0 & 0 & 1 \\
C & 1 & 0 & 0 & 1 \\
D & 0 & 1 & 1 & 0
\end{array}$$

Figure 1—Two graphs with same connection matrix

Were it not an important and confusing issue in computer graphics the preceding discussion might appear excessively lofty. But the semantic information that is important to CAGD distinguishes the area from computer graphics picture synthesis. In particular, there are many 3D configurations, some defying Newtonian physics, that project the appearance of a ball resting on a box. The *extent* (structural relationship) *of the model* is the same, but the *intent* (set of realizations) *of the model* is quite different. If the extent is the only concern, then they are all equally satisfactory, even though some configurations are not physically realizable; they look the same as one that is. The following repercussions of this issue will reinforce the importance of confronting this issue directly while the choice of model is still protean, before the implementation phase is begun.

As we have already observed in the example of the ball and the box, the choice of the rendering process for a model may critically affect the choice of extent for a model. A related example is in the use of shells or surfaces to represent solid 3D objects. For many applications this model is adequate, but as soon as we ask to examine a cross section of a solid using a surface representation we are confronted with the fact that this operation is not realizable until we augment the model with the notions of

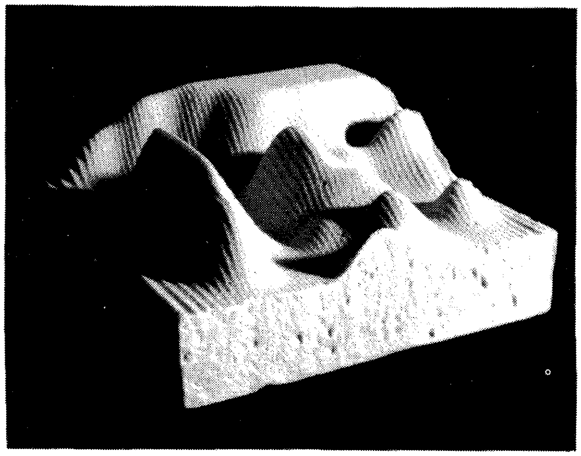


Figure 2—3D foam model (Courtesy of A. R. Forrest)

interior points and volumetric intersections. The analogous situation arises when we try to close a "clipped polygon."[26] Unless the polygon is thought of as a 2D object rather than a collection of lines and vertices, the notion of clipping a polygon is not well defined in the model. For these reasons projects like BDS[10] have adopted Baumgart's rather elaborate data structure.[3]

A CAGD system may rely on the rendering process instead of relying entirely on the geometric model, in which case the economy is occasionally called "cheating," because the model might not be rendered properly in another medium. This often occurs in *subtractive rendering* like numerically controlled milling or other machining operations, where important intersection curves are not explicitly computed. Rather they simply result from the successive subtractive operations. Recently there has been an active interest in curve[6,24] and surface descriptions that are entirely subtractive, that is, the curve is given by what remains inside an envelope of lines.

In still other situations we make use of an *additive rendering* process. Perhaps the most immediate example of this is in picture synthesis when a picture is "painted" into a large video memory, a frame buffer, by painting in the background first and then "adding in" objects that are closer to the eye. The "adding" takes place by adding together the transmittance coefficients associated with each object. Thus transparency and opacity are achieved through the additive rendering process.

Another example of additive rendering was part of an experimental effort by the Architecture Machine Group to build a responsive environment for a captive gerbil colony.[19] With the aid of a mechanical robotic hand, the computer actually stacked small building blocks. Thus the blocks "added" up to produce an environment.

The combination, an *algebraic rendering* process, presents itself as a rather different and generally unsolved problem. What parts of an assembly should be manufactured subtractively and then "added" together? These questions begin to border on the subject called computer aided manufacturing, and involve the problems of programming a soft assembly line that consists of machines, stations, materials, and programmable control.[27]

The attempted distinction between the model and the rendering process may seem somewhat turbid and belabored for the case of a single model visualized through a single rendering technique. When a single model is expected to suffice as the input for various rendering processes, then the distinction of what is model and what is part of the rendering process comes to the fore. For example, the surface shown is Figure 2 is a picture of a 3D model cut under computer control by a *3D plotter* developed at University of Cambridge. The actual model was a mathematical spline function, a B-spline surface[6,23] that was interactively designed. This plotter, however, differs from a 2D plotter in a few substantial ways. Besides the obvious restriction that it cannot cut a sphere in one setting of a block of material, we may inform some readers of the subtlety that instructions to a cutter normally refer to the position of the end of the chuck that holds the cutters.[4] The cutter cuts deeper than the position of the end of the chuck. Feeding the cutter the description in the model does not produce a proper rendition of the model. Rather, an *off-set surface* must be computed first before the 3D positions are passed along as instructions to the cutter. The off-set surface which is rendered then, is usually an approximation to the description in the model, as approximations are introduced in the off-set computation.

One apparent solution is to design the off-set surface rather than the surface itself. Sending that to the 3D plotter would produce exactly the desired output. A fault with this solution is that the designer probably wants the immediate visual feedback of a CRT, not the considerably delayed cutter response. We now have moved to the equivalent dual of the original problem. Only now the CRT instead of the 3D plotter requires the off-set description. Although it is dual computationally, it is less satisfactory from the point of view that, in making the model the off-set surface, we have confused the model with the rendering process. Imagine now a third rendering from an ordinary digital plotter, perhaps, with hidden lines removed. Or a shaded picture of the same model that might require a polygonal data base for the rendering process. Clearly we would not want to mill out the polygons. Nor do we have to remove hidden surfaces from a milled 3D model. On this point rests the case for separating model and rendering.

EXAMPLES

B-splines

A cubic *polynomial spline* is a curve that consists of cubic polynomial segments called *spans* which are joined together with continuous position, tangency and curvature. The spans of this *piecewise polynomial* are joined at the *knots* of the spline. A mathematical spline is an approximation to the behavior of a *mechanical spline,* a thin beam or elastica. The mathematical spline approximates the *minimum energy* form of an elastica.

Ordinary *cubic spline interpolation* involves calculating a cubic spline whose knots pass through some prescribed data points. The data points plus *continuity conditions* (position, tangency curvature) serve to specify the spline (Figure 3).

We can view this process as an interactive procedure for designing a smooth curve that meets preliminary interpolation constraints through control points {0, 1, . . . , 7} in Figure 3. The designer allows the design system to supply appropriate default parameters, thus producing a smooth curve passing through these designated points. In order to respond, the design system offers the unique curve that passes through the control points while satisfying the *variational criterion* of minimum ("linearized, total") curvature. Thus the system invokes a variational principle, thereby saving the designer from having to specify additional initial parameters.

Now suppose the designer is not satisfied with the initial response; he does not like the appearance of the curve in the region around data point 1 (Figure 3). Simply moving data point 1 and repeating the interpolation process, completely redefines every point on the curve. However, we only want to make a local change, not a global change, therefore, another procedure is necessary. There is no direct, simple, and predictable method for accomplishing a local change by varying the parameters of this *global scheme.*

Instead we invoke a *local scheme* for affecting the desired change, namely the *B-spline method*[16,23] which is a spline generalization of a technique for manipulating curves that was introduced by P. Bézier.[4] Now the control parameters for the curve are the vertices of its associated polygon whose shape the curve mimics (Figures 4 and 5). But each vertex has the special property that it only affects the curve locally. This is an example of different control parameters for the same geometric model. Manipulating the polygon permits a local alteration in a simple, direct operation, whereas manipulating the interpolation points did not. But the internal geometric model, the cubic spline, is the same regardless of the parameters used to specify it.

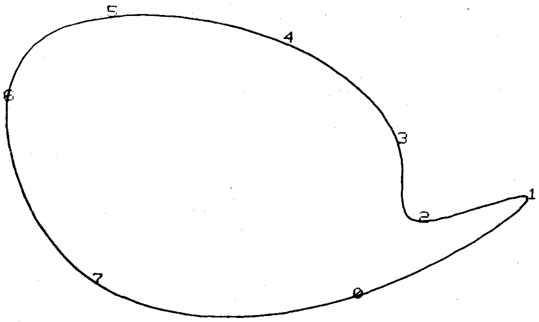


Figure 3—Interpolating periodic spline showing excessive bulge at point 1

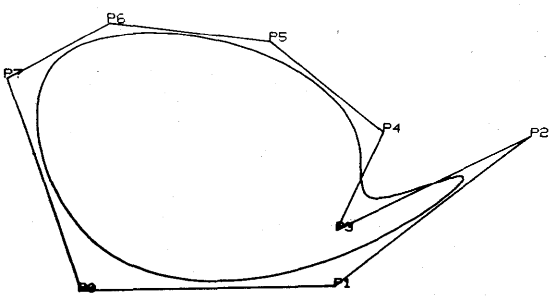


Figure 4—Curve from Figure 3 in B-spline form

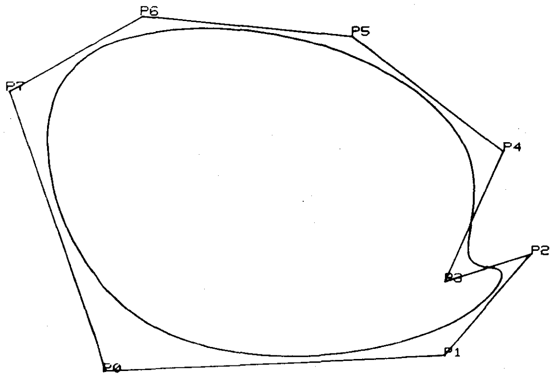


Figure 5—Curve from Figure 3 altered locally by moving vertex P2

On the other hand, if we expanded the model to *rational splines,* it would have increased the power of the geometric model. Rational splines represent circles exactly; cubic splines cannot.

Triangular patches

The most popular element for modelling a free-form surface has been the basic bicubic patch, proposed by Coons[7] and others (Figure 6). Being a bicubic map of the unit square, these patches are intrinsically rectilinear—they have four boundary curves. Although quilts of 4-sided patches are amenable to many situations, triangular patches are better suited to certain common situations

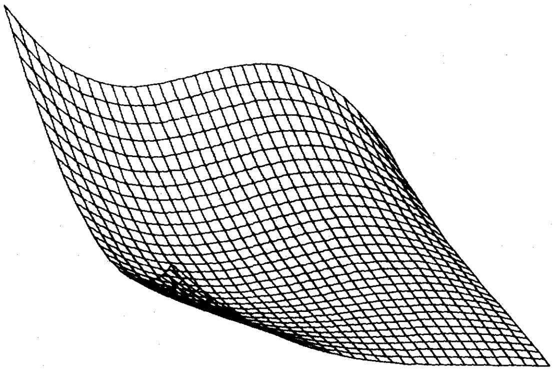


Figure 6—Bicubic Coons patch

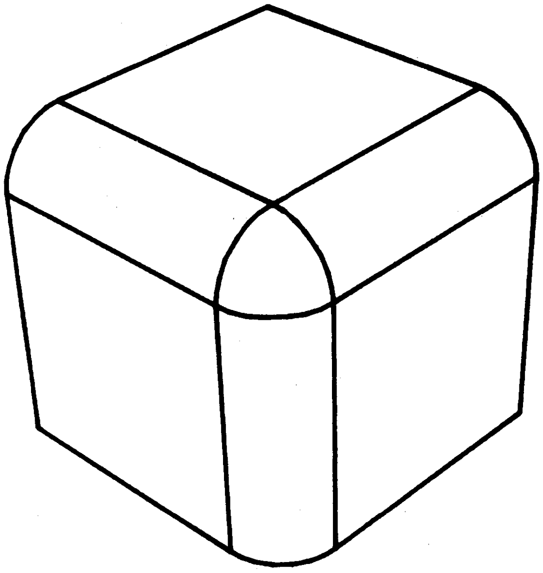


Figure 7—Curved triangular patch

(Figure 7). In this example, the 4-sided patch model does not permit the designer to specify triangular patches in a simple nondegenerate way. Recently Barnhill and Gregory[1,17] augmented the *Coons Patch model* with *rational triangular patches*. Barnhill and Gregory have had to expand the model to rational patches in order to include triangular elements as a proper mathematical extension.

CONCLUSION

In choosing a model for computer aided geometric design it is important to evaluate the utility of (1) the power of a model, and (2) the control parameters. Although they are intimately and inextricably related, a deficiency in one area is not necessarily remedied by an improvement in the other area. Furthermore, we conclude that the computer that houses a geometric model can also assist during the specification of a particular model by furnishing good default parameters and by calculating appropriate values of parameters that reflect constraints and relationships within the model.

ACKNOWLEDGMENT

Because the development of a viewpoint is an inscrutable, voluble process, I am not able to separate this acknowledgment into specific attributions of credit. Notwithstanding the inequity, may it suffice to express general recognition and appreciation to those many individuals who have influenced me on this subject. I thank R. McDermott for providing some of the figures, and making helpful suggestions during the writing of this paper.

REFERENCES

1. Barnhill, R. E., "Smooth Interpolation over Triangles," *Computer Aided Geometric Design* (Edited by Barnhill and Riesenfeld) Academic Press, 1974.
2. Baumgart, B., Ph.D. Thesis, Stanford University, 1974.
3. ________, "A Polyhedron Representation for Computer Vision," *These Proceedings*.
4. Bézier, P., *Numerical Control-Mathematics and Applications*. (translated by A. R. Forrest). London: John Wiley and Sons, 1973.
5. Braid, I. C., "Designing with Volumes," Ph.D. Thesis, Univ. of Cambridge, 1973. (Reprinted, Cantab Press, Cambridge, U.K., 1974).
6. Chaikin, G., "An Algorithm for High-Speed Curve Generation," *Computer Graphics and Image Processing,* June 1975.
7. Coons, S. A., "Surface Patches and B-spline Curves," *Computer Aided Geometric Design* (Edited by Barnhill and Riesenfeld) Academic Press, 1974.
8. Davis, P. J., *Interpolation and Approximation,* New York: Ginn-Blasidell, 1963.
9. Dube, R. Peter, *Local Schemes for Computer Aided Geometric Design,* Ph.D. Thesis at University of Utah, 1975.
10. Eastman, C., D. Fisher, G. Lafue, J. Lividini, D. Stoker and C. Yessios, *An Outline of the Building Description System,* Institute of Physical Planning Research Report #50. Carnegie-Mellon University (Sept. 1974).
11. Eastman, C., J. Lividini and D. Stoker, "A Database for Designing Large Physical Systems," *These Proceedings*.
12. Forrest, A. R., "Computer-Aided Design of Three-Dimensional Objects: A. Survey," *Proc. ACM/AICA Intern. Computing Sym.*, Venice, 1972.
13. Forrest, A. R., "Computational Geometry-Achievements and Problems," *Computer Aided Geometric Design* (Edited by Barnhill and Riesenfeld) Academic Press, 1974.
14. Freeman, H., "Computer Processing of Line-Drawing Images," *ACM Computing Surveys,* Vol. 6, No. 1, March 1974, pp. 57-97.
15. Gordon, W. J. and R. F. Riesenfeld, "Bernstein-Bézier Methods for the Computer-Aided Design of Free Form Curves and Surfaces," *J. of ACM,* Vol. 21, No. 2, April 1974, pp. 293-310.
16. Gordon, W. J. and R. F. Riesenfeld, "B-spline Curves and Surfaces," *Computer-Aided Geometric Design* (Edited by Barnhill and Riesenfeld) Academic Press, 1974.
17. Gregory, J. A., "Smooth Interpolation without Twist Constraints," *Computer Aided Geometric Design* (Edited by Barnhill and Riesenfeld) Academic Press, 1974.
18. Negroponte, N., "Graphics and architecture—Recent developments in sketch recognition," *Proc of NCC,* AFIPS Press, 1973.
19. Negroponte, N., *The Architecture Machine,* MIT Press, 1970.
20. Newell, M., *The Utilization of Procedural Models in Digital Image Synthesis,* Ph.D. Thesis at University of Utah, 1975.
21. Newman, W. and R. Sproull, *Principles of Interactive Computer Graphics,* McGraw Hill, 1973.
22. Resch, R. D., "The Topological Design of Sculptural and Architectural Systems," *AFIPS Conference Proceedings,* Vol. 42, AFIPS Press, 1973, pp. 643-650.
23. Riesenfeld, R. F., *Applications of B-spline Approximation to Geometric Problems of Computer Aided Design,* Ph.D. Thesis, Syracuse U., 1973. Available at Univ. of Utah UTEC-CSc-73-126.
24. Riesenfeld, R. F., "Note on Chaikin's Algorithm," *Computer Graphics and Image Processing* (to appear).
25. Sutherland, I. E., "Sketchpad: a man-machine graphical communication system," *Proc. SJCC,* 23, Spartan Books, 1963, pp. 329-346.
26. Sutherland, I. E. and G. W. Hodgman, "Re-entrant Polygon Clipping," *C of ACM,* Vol. 17, No. 1, Jan. 1974, pp. 32-42.
27. Voelcker, H., *Discrete Part Manufacturing: Theory and Practice,* TR of Production Automation Project, Univ. of Rochester, 1973.
28. Williams, R., "On the Application of Relational Data Structures in Computer Graphics," *Proc of IFIPS,* Stockholm. North Holland Publishing Company, Amsterdam; American Elsevier Publishing Company, New York, 1974.
29. *An Introduction to PADL,* Production Automation Project Staff TM-22, University of Rochester, December 1974.

A database for designing large physical systems*

by C. EASTMAN, J. LIVIDINI and D. STOKER
Carnegie-Mellon University
Pittsburgh, Pennsylvania

INTRODUCTION

To date, paper and pencil have been the principal medium for problem-solving and communication in architecture and engineering. While the computer has helped analyze topological and 2-dimensional problems, the drawingboard has remained the principal problem-solving tool for complex 3-dimensional systems.

Development of the Building Description System (BDS) was undertaken to explore use of the computer to construct models of complex physical systems. A fundamental goal of this work is a computer system capable of replacing drawings as the primary description for design and construction of buildings. The resulting capabilities are also directed toward the design of factories, bridges, mechanical systems, and urban planning.

BDS is based upon the assumption that an adequate description of a physical system is a specification of a set of elements together with their relative locations and that all other relations may be identified from these primitives. It provides facilities to define, modify, and arrange a large number of elements, coupled with a means to produce drawings of their arrangement and to analyze their performances. It also allows derivation and analysis of the spaces created by elements and automatic determination of spatial conflicts. The result is a single database useful for communication, analyses, coordination, and fabrication.

Elsewhere, several computer programs are under development that allow composition and analysis of a predefined set of building elements.[5,9] This restriction allows a priori analysis to identify all allowed conditions. Design, in their case, consists of composing elements so as to satisfy the predefined system constraints as well as desired building performances. An alternative approach, which we have followed, has been to develop a *general* description system, allowing definition or alteration of any possible element, and very general analytic and drawing routines.

In order to achieve these goals in a practical way, these performance requirements were further elaborated, to consist of:

(a) real-time manipulation of a database allowing the description of about 500,000 elements;
(b) easy definition and manipulation of arbitrarily complex shapes;
(c) a general facility for the association of attributes to shapes;
(d) structures for accessing elements according to their name, their properties, or their spatial location;
(e) implementation of the program in a readily available, inexpensive, computing environment.

In this paper, we present an overview of the database configuration of BDS which is designed to achieve these objectives. We shall particularly focus on: (1) the general facilities for describing an element in a concise manner and enabling easy definition and alteration, and (2) the organization of information about a very large number of elements, so as to allow real-time manipulation and accessing according to a variety of criteria. As will be seen, these two aspects are highly interrelated.

AN ENVIRONMENT FOR PHYSICAL SYSTEMS DESIGN

In the use of any large scale computer aided design (CAD) system, a *component library* of standard elements is expected to evolve to save users from repeated definition of commonly occurring components. During design, element descriptions will be transferred from it to a *project file* and augmented with information about location and performances. Accesses to an element description within the project file may be frequent, making efficiency of access important. Only one access of the library is necessary for each type of element, though, as copies can be easily made within the project file. If a custom element is to be reused later, it should be transferrable to the component library.

While a variety of hardware configurations could achieve these performances, our work has been designed for a 16-bit minicomputer (PDP-11/15) with 56K bytes of memory, supported by disc storage, a refresh graphics console[8] and interfaced with a large time-shared computer. The library is assumed to be stored on unmounted disc packs on the host or remote facility, while the project file will be maintained on-line on a disc file. All database operations will be made by

* This work has been supported by the National Science Foundation (GJ-42231X) and the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-73-C-0074).

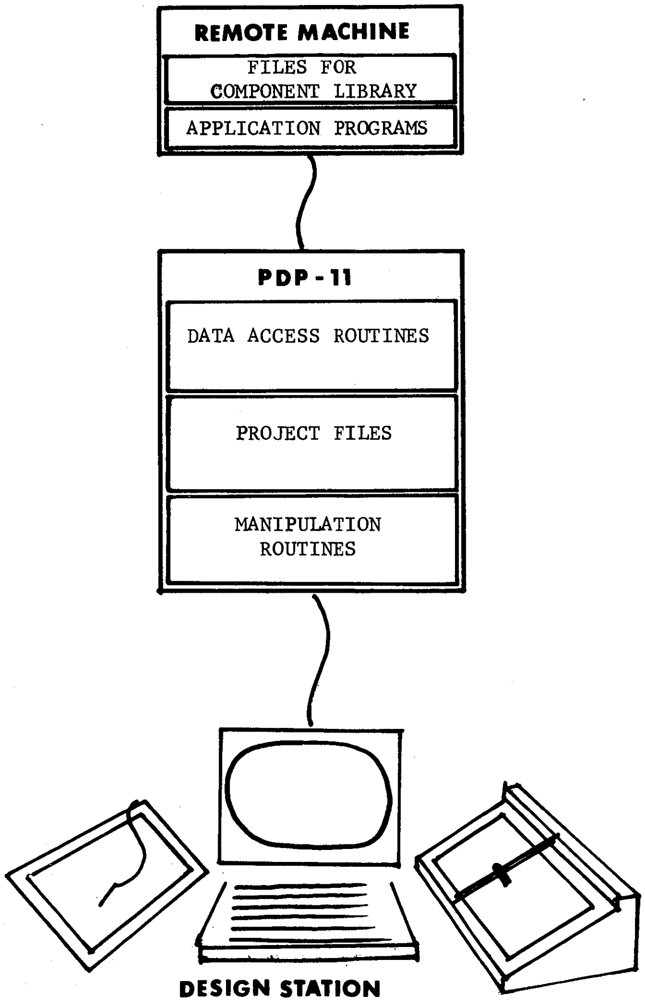


Figure 1—System organization of the building description system

the host minicomputer. The remotely connected machine is used for large application programs, interfacing with the database through a set of routines. This arrangement is schematically characterized in Figure 1.

REPRESENTATION OF AN ELEMENT

At the nucleus of the representation of any physical system is the information structure describing a single element. Most physical systems consist of an enormous set of different elements with complexity varying from a simple cube to a condensing tower, yet all design manipulations reduce to singular or compound operations upon the element data structure. Further, the performance of those manipulations are in part, dependent on the structure and content of the element information.

In general, any representation intended for machine aided design should respond to three general requirements:

1. the structure should be adequately *general* to enable storing the description of elements of vastly different complexity;
2. the structure should be sufficiently "rich" to facilitate the operations to be applied to it, and to define the range of properties of interest. Among the properties desired in this case are capabilities for an open-ended set of attributes, identification of inside-outside, and operations for manipulating element descriptions, including shape and location;
3. the structure should be efficient, both in terms of the space required to store an element description, and in terms of the time required to manipulate it.

Realization of these requirements is facilitated by introducing a level of abstraction to the concept of an element description. An element description may be thought of as three separate parts, as a *topology*, a *geometry*, and a set of attributes. Thus, a cube might be thought of as a topology of six four-sided, inter-connected planar surfaces, a geometry made up of eight coordinate triples locating the eight corners, and a set of attributes, such as surface texture, composition, weight, etc.

This abstraction provides several distinct advantages in satisfying the above criteria. Operations on an element consist of distinct actions on these three parts. The spatial transformations (rotation, translations, scaling) act on the geometry, as do the perspective and other visual transformations. These operations are well defined and efficient algorithms exist to perform them.[7]

The Euler Operations

Operation	Equation (faces) −	(edges) +	(vertexes) =	(2*objects) −	(2*holes)
Construct edge and vertex		+1	+1		
Construct face and edge	+1	+1			
Construct object and face, destroy edge	+1	−1		+2*1	
Split an edge with a vertex		+1	+1		
Divide an object with a plane	+2			+2*1	
Fuse two objects and remove the common faces	−2			−2*1	
Remove two faces and construct a hole	−2				+2*1

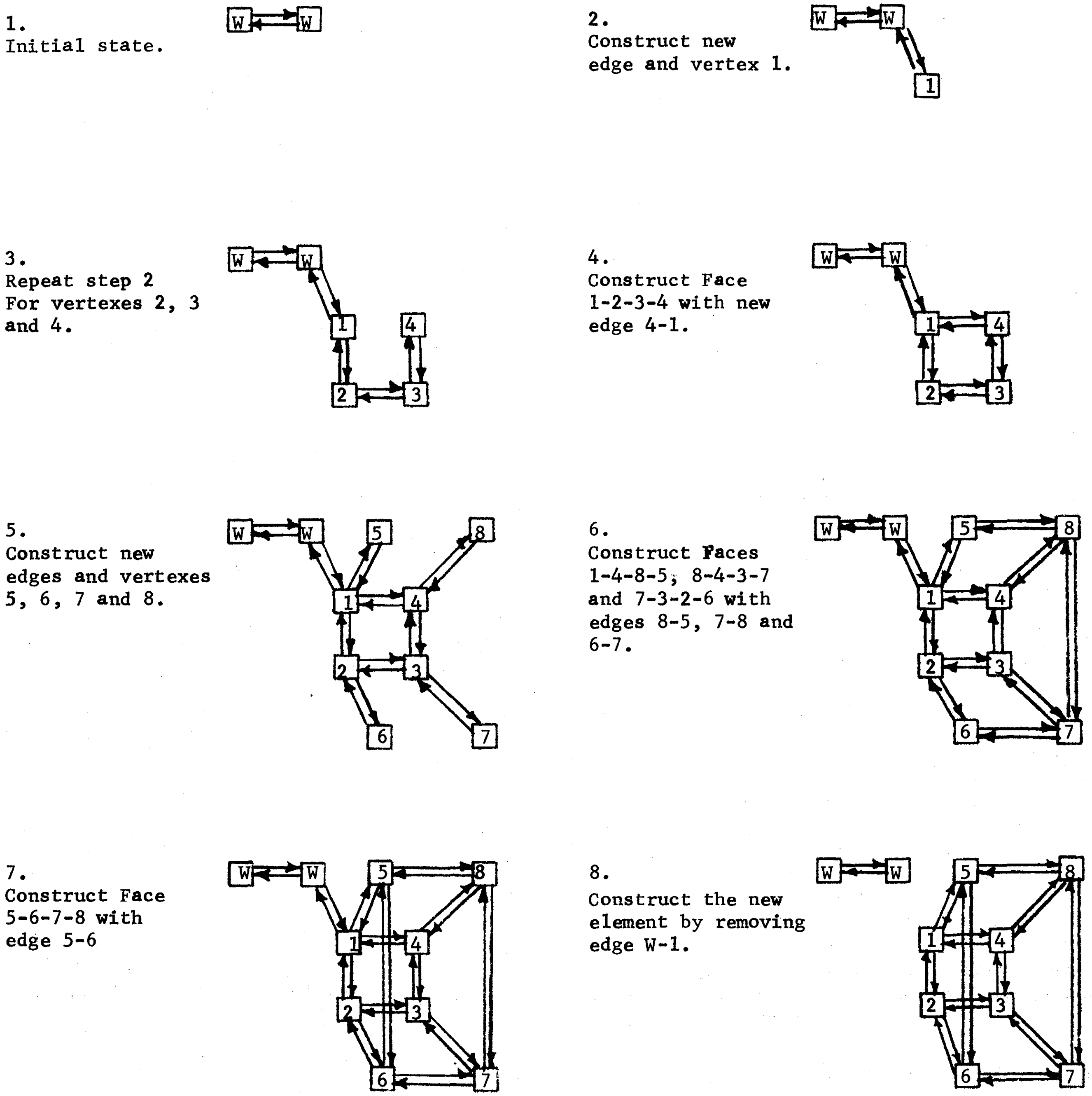
Figure 2—Definition of the Euler operations and their consistency with regard to Euler's equation

As Baumgart has pointed out,[1] operations on topologies are also well defined. The equation first proved by Euler:

$$(\text{faces}) - (\text{edges}) + (\text{vertexes}) = 2^*(\text{bodies}) - 2^*(\text{holes})$$

provides not only the fundamental canonical relations within all polyhedral topologies, but also suggests a discipline for defining the operations needed to perform any topological construction or manipulation. A set of operations, called the Euler operators, are enumerated in Figure 2. Each operation structurally alters the topology to which it is applied, but in such a way that the Euler equation is satisfied. Thus each operation results in a well-formed element, eliminating problems of ambiguous line or coordinate definition due to an incomplete element description. Figure 3 illustrates how these operators may be used to construct a cube. They may be used to construct or alter any polyhedral topology.

Other topological operations can be expressed in terms of the Euler operators. For example, using both topological and



1. Initial state.

2. Construct new edge and vertex 1.

3. Repeat step 2 For vertexes 2, 3 and 4.

4. Construct Face 1-2-3-4 with new edge 4-1.

5. Construct new edges and vertexes 5, 6, 7 and 8.

6. Construct Faces 1-4-8-5; 8-4-3-7 and 7-3-2-6 with edges 8-5, 7-8 and 6-7.

7. Construct Face 5-6-7-8 with edge 5-6

8. Construct the new element by removing edge W-1.

Figure 3—An example of the construction of a topology using the Euler operators

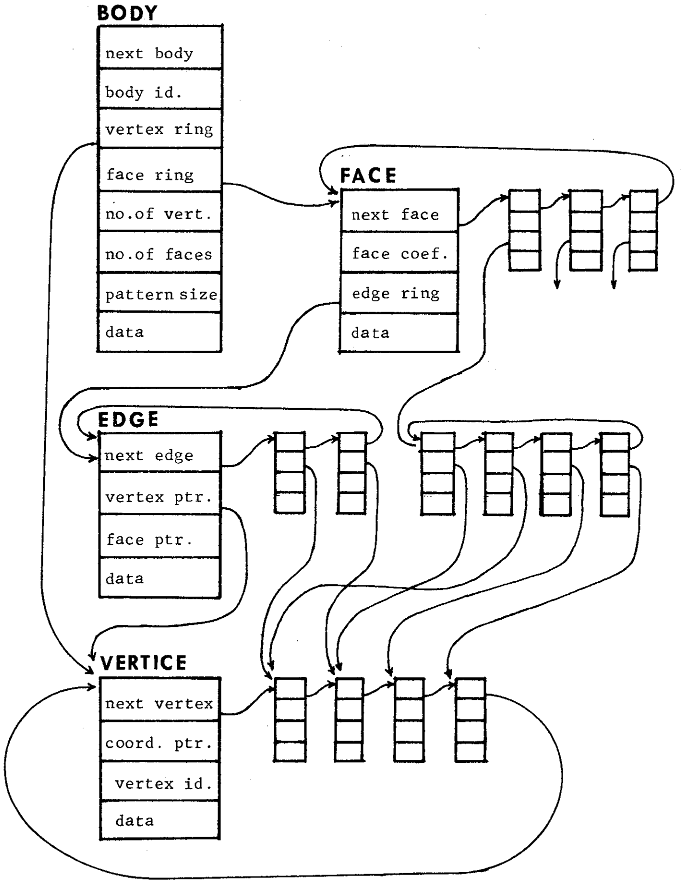


Figure 4—Data structure for depicting the topology of an element shape

geometric primitives, we have implemented the set operations (union, intersection, differences). The set operations provide the necessary primitives for manipulating the shapes of elements and for testing spatial conflicts between them.

The analysis and manipulation of attribute information is usually treated by various types of application programs. Operations on attribute information will not be further developed here.

The three parts to an element description each require their own data structure. The topology data structure, upon which the Euler Operators apply, is shown in Figure 4. The "clockwise ordering" of the vertexes defining an object face (planar surface), may be used to easily determine the relative location of a point.[1] Adjacency relationships between element faces are recorded and the notion that a topological "edge" represents the boundary between exactly two faces is maintained. Also properties (texture, reflectance) may be assigned to a face. The geometry data structure, in its most complete form, is simply a vector of coordinate values. The attribute data structure is described later.

HIERARCHICAL DEFINITION OF SHAPE

A major benefit of separating the topology of a shape from its geometry is the efficiency of storage gained. One topology may be common to a wide variety of shapes; a rectangle, for instance, defines the shape of all doors, bookshelves, windows, and structural timber within a building. Similarly, the topologies for all steel WF members are the same. Conversely, multiple topologies utilizing a common set of coordinates are unlikely ever to occur (though they can be conceived). The relation between topologies and geometries suggests a *hierarchy* of shape description, with the geometries of different elements sharing a single topology to describe a set of elements. Thus the cost of a topological description can be spread over the number of geometrical descriptions which access it.

A further saving is possible by recognizing that in most areas of design—and particularly building design—many elements are used repetitively. Instead of describing each geometry separately in a common base coordinate system, each element can be described in local coordinates and mapped via a spatial transform into world coordinates. Only a separate transform is needed to generate the unique coordinates of an element. This introduces a third level to the database hierarchy. The hierarchy consists of topology, local coordinate and transform levels.

Each level in the hierarchy defines a class of information which is shared by all elements below it. By describing the common information once, redundancy is reduced in the database.

Other redundancies can also be identified. One pertains to

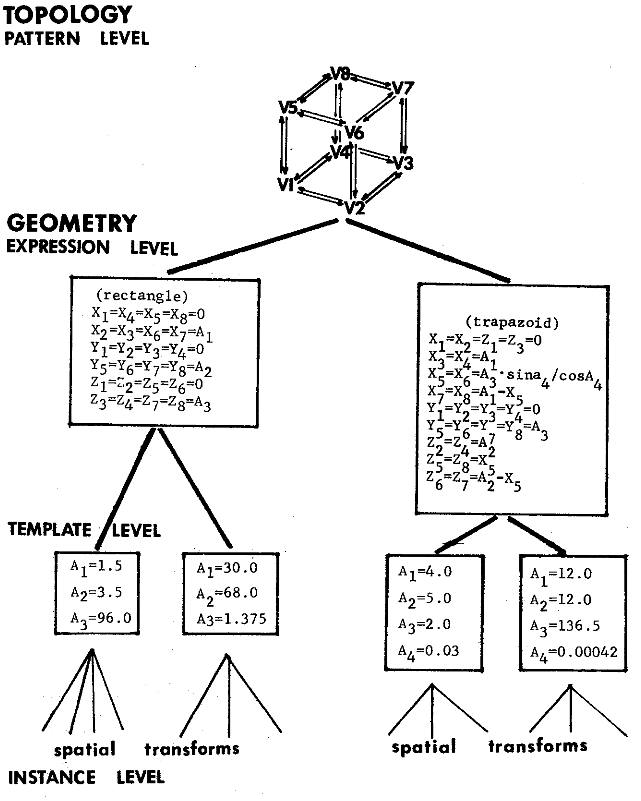


Figure 5—The four hierarchical levels within the BDS database. The shapes depicted are an eight foot 2×4, a door, a rubber motor mount, and a concrete column respectively

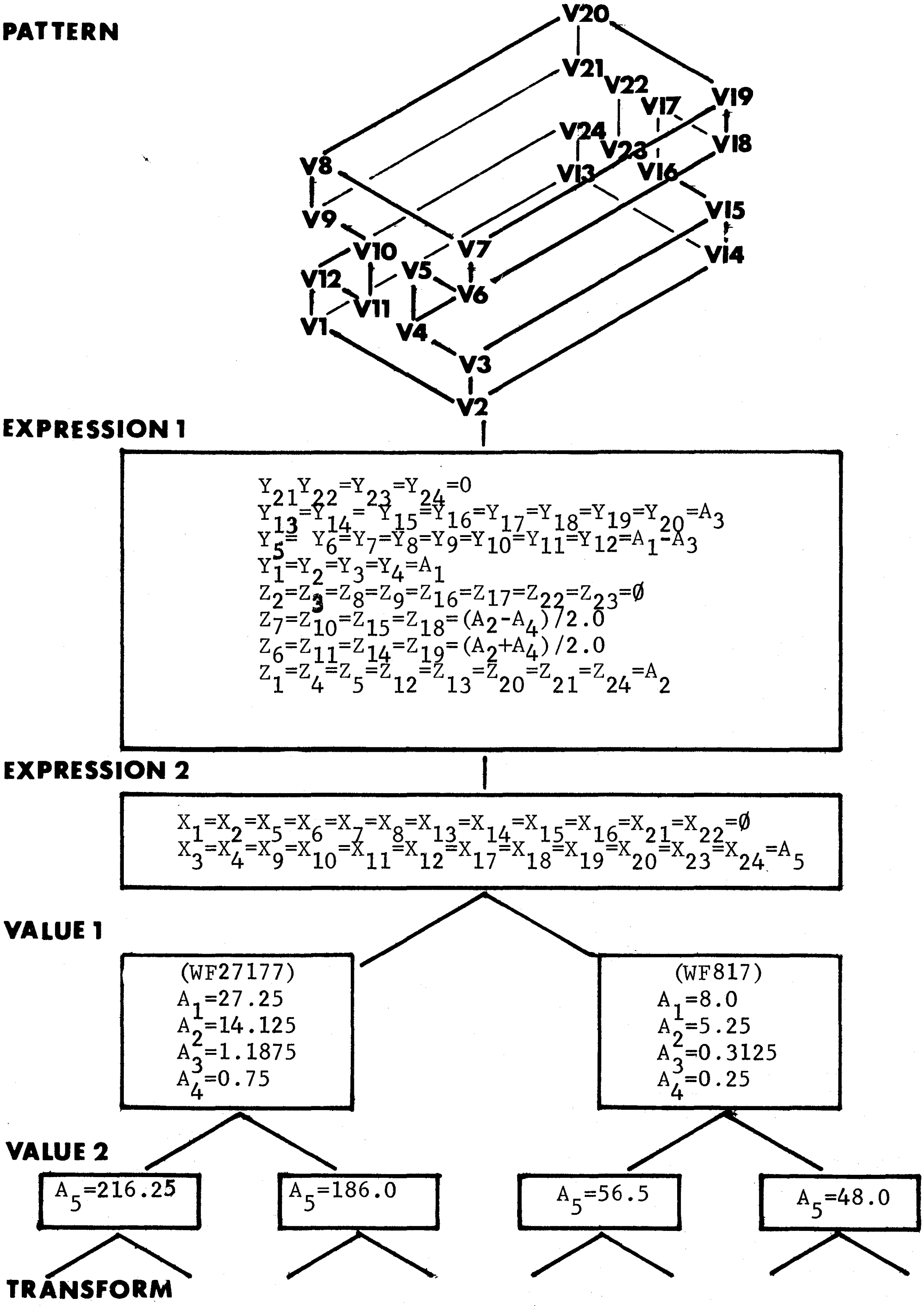

PATTERN
V20
V21
V22
V17
V19
V24
V23
V18
V8
V13
V16
V9
V15
V10
V7
V14
V12
V5
V6
V11
V1
V4
V3
V2
EXPRESSION 1
$Y_{21}Y_{22}=Y_{23}=Y_{24}=0$
$Y_{13}=Y_{14}= Y_{15}=Y_{16}=Y_{17}=Y_{18}=Y_{19}=Y_{20}=A_3$
$Y_5= Y_6=Y_7=Y_8=Y_9=Y_{10}=Y_{11}=Y_{12}=A_1-A_3$
$Y_1=Y_2=Y_3=Y_4=A_1$
$Z_2=Z_3=Z_8=Z_9=Z_{16}=Z_{17}=Z_{22}=Z_{23}=\emptyset$
$Z_7=Z_{10}=Z_{15}=Z_{18}=(A_2-A_4)/2.0$
$Z_6=Z_{11}=Z_{14}=Z_{19}=(A_2+A_4)/2.0$
$Z_1=Z_4=Z_5=Z_{12}=Z_{13}=Z_{20}=Z_{21}=Z_{24}=A_2$
EXPRESSION 2
$X_1=X_2=X_5=X_6=X_7=X_8=X_{13}=X_{14}=X_{15}=X_{16}=X_{21}=X_{22}=\emptyset$
$X_3=X_4=X_9=X_{10}=X_{11}=X_{12}=X_{17}=X_{18}=X_{19}=X_{20}=X_{23}=X_{24}=A_5$
VALUE 1
(WF27177)
$A_1=27.25$
$A_2=14.125$
$A_3=1.1875$
$A_4=0.75$
(WF817)
$A_1=8.0$
$A_2=5.25$
$A_3=0.3125$
$A_4=0.25$
VALUE 2
$A_5=216.25$
$A_5=186.0$
$A_5=56.5$
$A_5=48.0$
TRANSFORM

Figure 6—The final six level instantiation hierarchy implemented in BDS

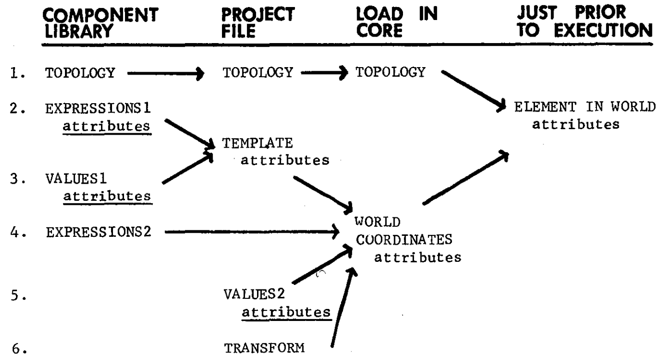


Figure 7—Computation sequence in instantiating an element. Those attributes underlined are stored, the others computed

common *families of shapes*. The number of vertex coordinates required by a shape is usually much greater than the number of parameters required to describe it. Any rectangle is defined by three parameters. In the structural steel handbooks, the shape of all WF elements are defined in terms of four parameters, depth, width, flange thickness and web thickness. It is straightforward to define a set of expressions that converts the essential shape parameters into the corresponding set of vertex coordinates. See Figure 5. Expressions may be derived that convert a set of parameters into vertex coordinates for any shape based on a single topology. The resulting savings are significant. One pattern and set of expressions, together with a collection of parameter sets, can now, for example, represent all rectangles or WF beams within a complex design.

In some cases, we have found it worthwhile to partition the expressions and parameters even further. Those dimensions fixed by the selection of the element are derived first, by applying the appropriate parameters to a first set of expressions. This first set defines the section of extruded or the thickness of rolled stock. Thus all elements made of the same stock can use common expressions and values. Later, individual lengths or sizes can be derived by applying a second set of values to a second set of expressions, deriving the rest of the shape coordinates. A path through the six level hierarchy describes the shape of any particular element. An example of the information stored in the hierarchy is shown in Figure 6.

Of course, all these levels of description need not be used if there are no corresponding variations within the element set being described. For the concise description of elements used in building, though, we have found it efficient to use all six levels for standard, predefined elements and the three initial levels of topology, geometry, and transform for custom fabricated elements.

Significant storage efficiencies are gained through the hierarchical storage of shapes. A cost has been imposed, though, in the computation required to create a full shape description. An overall benefit is achieved only if the computations to create a full description are inconspicuous to the user. The computation sequence used in BDS is shown in Figure 7. Most of the computation is undertaken when the element description is transferred from the component library to the project file. All other computations, except assignment of the base address for the coordinates, are done when the element is loaded into core for operations. The analysis leading to this particular implementation has been described elsewhere.[4]

ATTRIBUTE INFORMATION

In general, attributes are needed in CAD to describe such properties as an element's manufacturer, cost, function, performance, and other properties. Unfortunately, the range of possibly relevant properties is too large for a single set of predefined attributes to be practical.

Among the criteria important in the definition of attributes are: (1) the capability to associate attributes with specific elements as well as with any arbitrary collection; (2) the ability to introduce or modify attribute names and values at any time during design.

Considered most generally, an attribute consists of three pieces of information: a name, a data type, and a value. We have found the need to include three different data types: Number (on input, integers are converted to real); Character string, and Set. All attribute names are coded in an extendible dictionary; so are all character strings.

The element hierarchy can be used to advantage, allowing the storing of attribute triples at multiple levels. For example, the name "manufacturer" and its character string value can be associated with the topology; the name "cost" and a numerical value can be associated with the template. See Figure 8.

The element hierarchy does not provide a mechanism for assessing elements by function and an alternative hierarchical scheme is needed. An attribute type called *Set* may be used to construct an inverted file based upon any common attribute values at any level, as shown in Figure 8. The file can be assessed in a variety of ways and can be made up of other sets, making a set hierarchy.

SPATIAL ORIENTED ACCESSES

The retrieval method for any part of the hierarchical data structure is through a Directory within the project file. Elements with similar properties are accessed through the

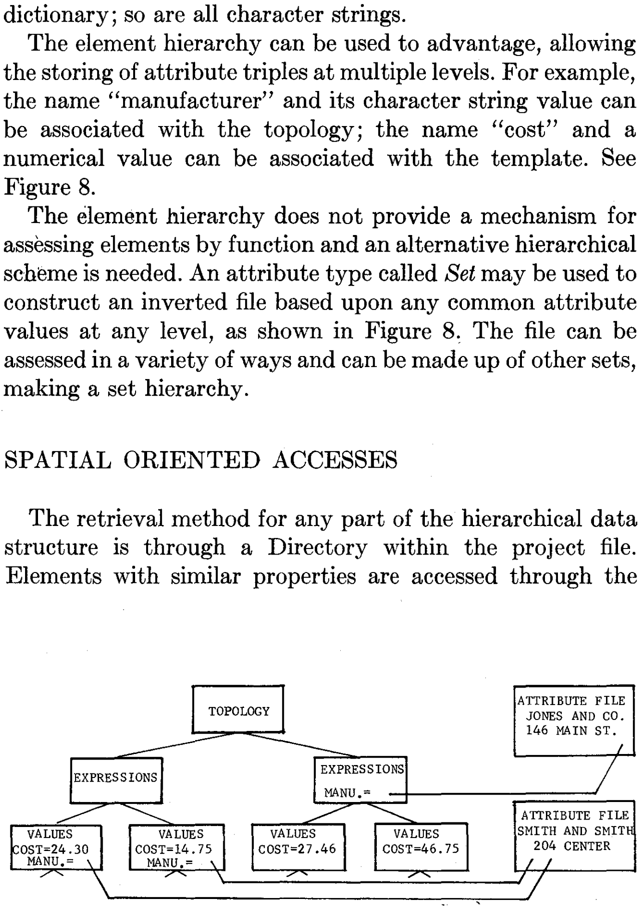
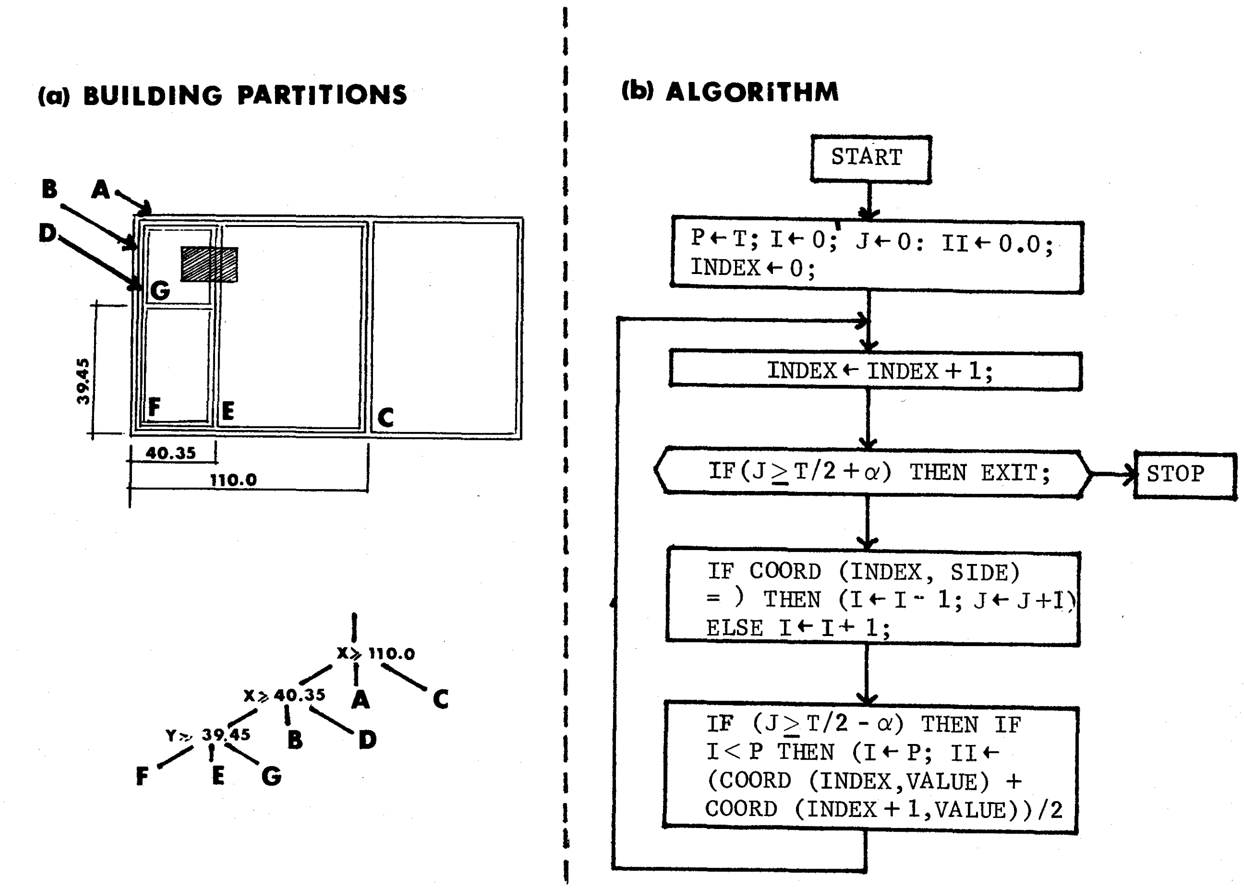


Figure 8—Facilities for storing and accessing attributes. Cost is stored as a real value within each template, whereas manufacturer is a type SET allowing accesses to all elements with the same value


(a) BUILDING PARTITIONS
B
A
D
G
F
E
C
39.45
40.35
110.0
X≥110.0
X≥40.35
A
C
Y≥39.45
B
D
F
E
G
(b) ALGORITHM
START
P←T; I←0; J←0: II←0.0; INDEX←0;
INDEX←INDEX+1;
IF(J≥T/2+α) THEN EXIT;
STOP
IF COORD (INDEX, SIDE) =) THEN (I←I−1; J←J+1) ELSE I←I+1;
IF (J≥T/2−α) THEN IF I<P THEN (I←P; II← (COORD (INDEX,VALUE) + COORD (INDEX+1,VALUE))/2

(c) EXAMPLE

COORD (VALJE, INDEX, SIZE) where EXAMPLE

INDEX :: = 0 ≤ integer ≤ (T*2)	1	2	3	4	5	6	7	8	9	10	11	12
VALUE :: = real number	2.0	5.5	6.5	7.0	7.8	8.4	8.4	9.2	10.0	10.2	11.6	13.6
SIDE :: = (for min and) for max	(1	(2	(6	(7	1)	6)	(5	(8	2)	8)	7)	5)
ALGORITHM I	1	2	3	4	3	2	3	4	3	2		
RESULTS: II	0	0	0	0	8.2	8.4	8	8	10.1	10.4		
J	0	0	0	1	2	2	2	2	3	4		
P	6.0	6.0	6.0	6.0	3.0	2.0	2.0	2.0	2.0	2.0		

Figure 9—A method for determining the spatial partition of a full block of elements

α: : = the range from middle of allowed partitions (a tuning parameter)

I: : = number of elements cut by partition

II: : = coordinate value of next partition

J: : = number of elements to the left of current pointer

Set values, which are also stored in a Directory. A typical requirement for CAD systems, though, is to access collections of spatially related elements. This involves finding all elements within the data structure which, when transformed into the world coordinates, occupy or spatially conflict with a specified volume of space. None of the structures described thus far collect together elements which are spatially related and an exhaustive search would be prohibitive.

To access element information spatially in an acceptable amount of time, it is necessary to eliminate without inspection as many elements as possible which might prove to be completely disjoint from the search volume. A preliminary search can eliminate those elements whose envelopes are disjoint with the envelope of the search volume. By an *envelope*, we mean the minimal rectangle enclosing an element, with the rectangle's faces parallel to the coordinate axes. This initial search would hopefully generate a small enough set of elements that can be tested exhaustively.

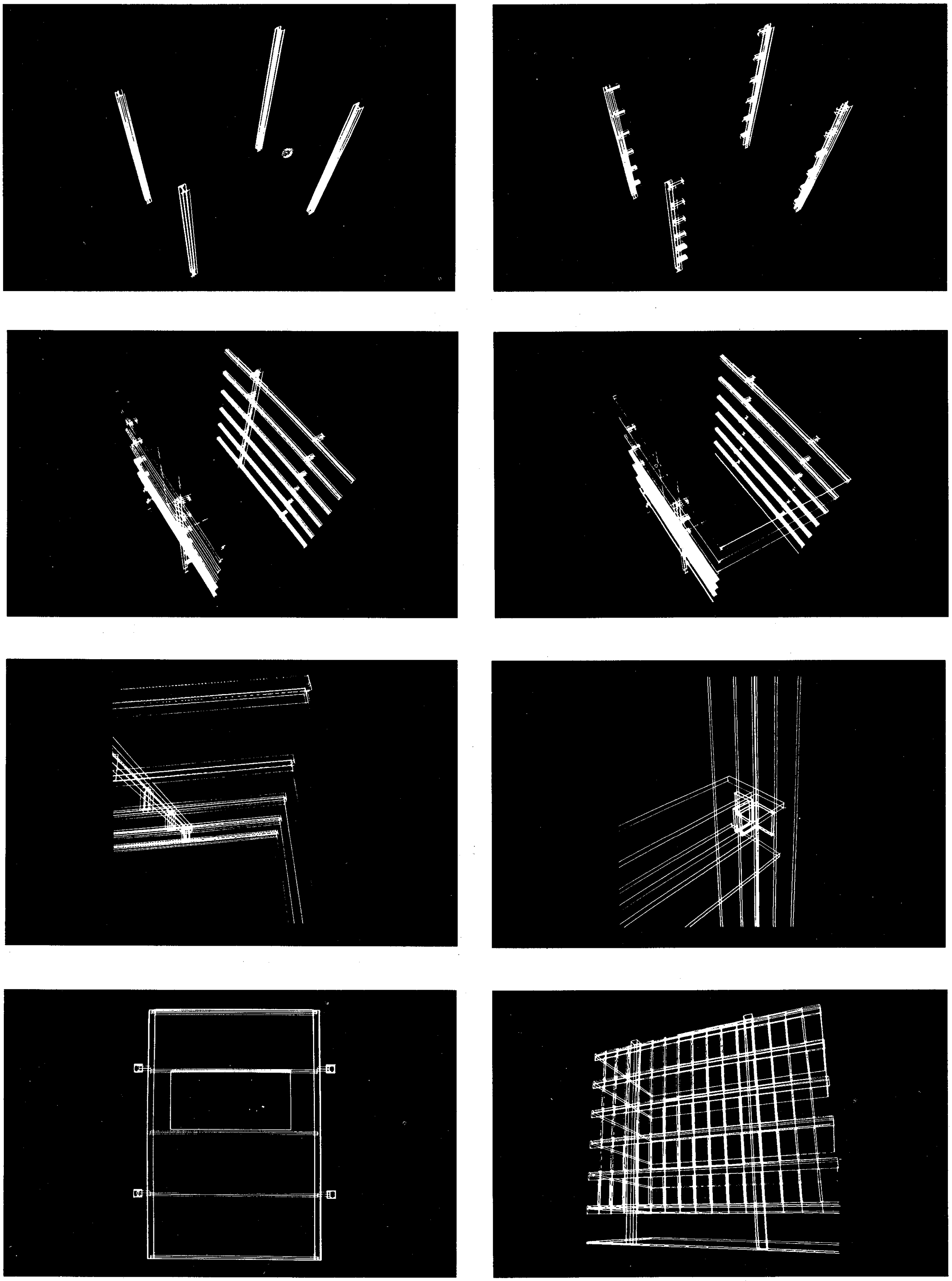


Figure 10

S. Leinhardt

Rather then attempt to superimpose on the hierarchical data structure an accessing mechanism which would allow this search, it is much more efficient to manage element allocations within secondary storage on a spatial basis and set up a tree structure for accesses. See Figure 9(a). The envelopes of elements will be located together within some type of block structure, e.g., disc segments, each block corresponding to a rectangular area within the design space. When a block becomes full, the elements within it can be ordered by their envelope coordinates along the longest dimension of the rectangular volume. By applying the algorithm sketched in Figure 9(b) to the coordinates, they are partitioned into two approximately equal sets by a plane through the rectangular volume resulting in the fewest overlaps with the elements. Those elements which do overlap are allocated to a third block. As this partitioning takes place, the access tree is modified accordingly. With this structure a spatial search involves scanning only the search tree to identify all those blocks overlapping the volume of interest. Once this is done it is necessary to test exhaustively the elements within these partitions for conflicts.

The maximum size of an element within a block cannot be larger than the block dimensions. This accessing scheme thus proves useful in selecting elements by size for display. This organization and search scheme evolved from one developed during the CEDAR project[9] and is more fully described elsewhere.[4]

CONCLUSION

At some future point in time, we predict that most design and drafting will be undertaken in a system environment quite different than that which exists today. Design will be done at a CRT console, as first imagined by Coons[2] and Licklider.[6] The designer will interactively construct a large database representing his design. He may interactively manipulate and analyze, or automatically generate subsystems. For analyses, general purpose analysis programs will rely on a set of interface procedures that will access the data needed. Elements may be automatically selected by the application programs.

A project database will be backed up by a large library of standard elements, readily available for machine accessing in a compatible format. These parts libraries will reside locally or be remotely accessible and maintained by a service bureau or similar organization. Custom elements will be easily added by a mixture of graphic and typed input. Members of the design team may be remotely located, but all working on the same system description. Each project will reside on a mass storage device, most likely a disc pack. As the design evolves, cost estimates, bills of materials, and construction schedules will be prepared, by a single organization or by many different ones using a common database. Building or other legal codes will be just another form of analysis program. Contractors or other fabricators will use the database produced during design for parts ordering and analysis of the construction sequence. For some parts, it may allow automated fabrication. Later, the database could be used for monitoring operations, maintenance, and depreciation on the elements or the system as a whole.

BDS was conceived as a prototype for developing the techniques needed to realize *general* building description systems. Most of its features are operational and have been used to design several simple buildings. Some of the graphic output available for one of the buildings is shown in Figure 10.

BDS was written in BLISS, a system building language developed at C-MU and operates in the PDP-10 as well as PDP-11. The program size currently 120K bytes. Database capacity is a function of disc storage available. The average density is approximately sixty bytes per element.

BIBLIOGRAPHY

1. Baumgart, B., *Winged Edge Polyhedron Representation*, Stanford A.I. Project Report AIM-179, October 1972, Stanford University.
2. Coons, S. A., "An Outline of the Requirements for a Computer-Aided Design System," *Proceedings Spring Joint Computer Conference*, Spartan Books, Washington, D.C. 1963.
3. Eastman, C. and J. Lividini, *Spatial Search*, Institute of Physical Planning Research Report, Carnegie-Mellon University, September 1974.
4. Eastman, C., J. Lividini and D. Stoker, *A Database for Very Large Physical Systems*, Institute of Physical Planning Research Report, Carnegie-Mellon University, February 1975.
5. Hoskins, E. M., "Computer Aids in Building," in *Computer Aided Design*, J. J. Vlietstra and R. F. Wielinga (eds.) American Elsevier, New York 1973.
6. Licklider, J. C. R., "Man Computer Symbiosis," Institute of Radio Engineers on Human Factors in *Electronics*, Volume HFE-1 #7, March 1960.
7. Newman, Wm. and R. Sproul, *Principles of Interactive Computer Graphics*, McGraw-Hill, New York 1973.
8. Rosen, Brian, "The Architecture of a High-Performance Graphics Display Terminal," *1973 Society for Information Display International Symposium*, May 1973, New York City.
9. Sampson, Peter, *The Implementation of CEDAR: A Computer-Aided Building Design System*, unpublished report Design Group, Royal College of Art, London, August 1973.

Economic principles for interactive graphic applications

by S. H. CHASEN
Lockheed Aircraft Corporation
Marietta, Georgia

BACKGROUND

With the continuing national economic malaise, there is a steadily increasing consciousness of the need to understand cost/benefits relationships which will result from the use of additional or alternative technical tools. This is particularly true in the consideration of interactive computer graphics (ICG) as a viable and valuable adjunct to our technical options. Although much potential exists to enhance our understanding of the economics of ICG, very little hard data has been forthcoming for a variety of reasons. There are a number of attributes which have been applied to ICG and which represent implied or intangible value. The oft-stated advantages of lead time reduction, reduction of the number of evaluation cycles, improved product quality, better training and comprehension, etc., have various degrees of economic benefit equivalence, but such benefits are difficult to quantify and they depend heavily on both the nature of the application and of the graphic user. For example, if time savings are manifest for tasks which lie on or close to the "critical path" of a multi-task project, cost benefits as a function of time saved might be easily verified and easily computed. On the other hand, savings of time on non "critical path" tasks may have little or no economic value whatsoever. Generally, however, the attributes stated above imply economic advantages on primarily qualitative terms. The problem of determining economic benefits is exacerbated by the method by which graphic facilities are charged. Prorated costs for additional systems and support are quite varied from institution to institution and opportunities for a variety of lease or buy options will impact the computation of cost effectiveness. It should be recognized that an appropriate console use charge for ICG most decidedly impacts ICG utilization and the level of utilization feeds back to affect the charge. Thus, early potential users of a start-up and low-use ICG system may be deterred for budgetary reasons. Therefore, maximum development of cost-effective operations would be possible if ICG use charges were predicted on an "expected" level of use over a reasonable time frame. An appropriate charge can be established as follows. First, determine use charge based on use of graphics of all applications where graphics could show any benefits potential. The resulting charge would then be based on a maximum operation of the system. This charge would then be studied in light of the prospective potential applications to eliminate those for which this "minimum" charge would be too great for cost effective ICG use. With the smaller set of potential applications, an amended use charge would be computed, remaining applications would be studied in light of the modified charge, and the iterative cycle would be continued until there is convergence on charges and applications which will use the system. If there is no convergence, then there would be no application set for which graphics is cost effective. The charge should be subsidized or put into overhead initially to give ample opportunities for ICG applications to reach their potential. Subsequently, the. charge to users would be periodically reviewed and modified. This approach to ICG charges can yield the optimum number of graphic scopes. This will be described later in more detail.

Another problem in determining cost effectiveness or benefits for ICG is characteristic of people and of their organizations. In general there seems to be little or no funding mechanism to make serious studies and to perform needed controlled comparative experimentation to derive hard data to clearly set forth the advantages of ICG. Without such a mechanism there is no eagerness on the part of each potential user to spend his own budget to perform modest research on the cost effectiveness for his application. Where good benefits data have been developed, most ICG users have no particular motivation to publicize it. Thus, we note the continued expanded use of ICG (which implies cost benefits), but good factual data is very difficult to find. The most prevalent data have been developed using the hard data liberally augmented by the "educated" estimates of experts in various application areas for ICG. The thoroughness, the method of assessment, and indeed the reliability of such estimates will vary among users. The important point is that most institutions come out with similar benefit expectations for similar applications.

In spite of the lack of large quantities of accessible hard data on ICG economics, there have been some data developed in the past, some data is being developed, and there is a considerable amount of unexploited potential in this area. In 1969, for example, the Lockheed-Georgia Company documented a savings of $250,000 for the first nine months of use of the 3-scope IBM 2250 and 360/50 system as applied to 2-D structural analysis. This cost reduction was recorded as Lockheed Report CRR#68-12-26G-01 under the DOD cost reduction program. It was a net value after subtracting appropriate costs for additional hardware, for system maintenance, and for system implementation. It did not reflect earlier costs for research and development. It is interesting to note that the largest portion of cost reduction was achieved in the data evaluation and documentation phases. Net computer costs, including graphics charges, were increased, although that may not necessarily be the case for certain other applications.

The well-publicized numerical control part programming application at Lockheed-Georgia did not improve manpower productivity as much as initial estimates of six to one had predicted. The actual increase of productivity by a factor of three (one man hour at the graphic console would produce the equivalence of three man hours of conventional N/C work) was estimated to defray the additional costs associated with the utilization of graphics for N/C programming. The other benefits of time, product quality, better utilization of scarce manpower skills, etc., tipped the scales in favor of graphics for N/C part programming. The use of the Lockheed-Burbank developed CADAM, Computer Augmented Design and Manufacturing, further enhanced the value of N/C graphics since CADAM produces the geometry (on which N/C cutter paths are derived) at an earlier stage of design—in the detailed drafting phase via graphics. CADAM itself has received very heavy use at both Lockheed-Burbank and Lockheed-Sunnyvale. It increases productivity over a range of values depending on the application type. Its heavy utilization and a Lockheed-Burbank software development which operates many terminals within a single core partition have teamed to lower costs to operate graphic consoles and to make CADAM cost effective. Efforts to reduce hardware costs through consideration of alternative displays and of mini computers promise to further enhance CADAM for design drafting, N/C, and other applications.

There are many relatively simple tests and analyses that could be assessed, in the light of potential applications, that would indicate the potential economics of various graphic system options. For applications which require analysis, multiple parameters, iterations, and plotted results, it may suffice to affix a direct view storage display directly to a time share system. It should be very easy, in most cases, to show cost effectiveness with this very low cost addition. Of course, the level of sophistication of interactive graphics that will produce the maximum benefit is a function of many factors and is, therefore, difficult to determine. Among the ICG options are: stand alone single display systems, stand alone multiple display systems, multiple display and other consoles within a stand alone system, non-stand alone host/satellite ICG systems, and systems that can operate either in stand alone or host/satellite mode. In most modern and future options of this sort, mini computers will play a prominent role. Thus, mini computer characteristics, display type and facilities, communication options, peripheral devices, applications mix, available software, organizational considerations, and many other factors must be weighed in a veritable infinitude of combinations and permutations.

Although costs are continually falling for both hardware and software, it will be asserted without proof that most ICG applications of substance will require a graphic console and peripherals ranging in equivalent purchase price from approximately $75K to about $200K and even higher (excluding the case of the direct linkage between the computer and a storage tube described earlier). Of course, multiple consoles can be implemented at lower costs per console, in general. (Special CPU requirements and system support will increase the unit cost, as is the case for CADAM.) Although such costs would be contested by some, the figures are intended as more or less an all-inclusive range and are an attempt to put vastly differing systems on a common basis.

To interpret this range in cost/hour for the system, one may consider a variety of lease/buy/accounting alternatives. These options will not be discussed. However, nominal values will be assumed here to put matters into perspective. Suppose, for example, the ICG system is to be amortized over five years. The number of work hours in a standard work year is taken, for convenience, to be 2000 hours. Thus, amortization is taken, in this example, over a period of 10,000 hours. This translates the console purchase price range from above to $7.50 through $20 per hour for a standard eight hour work day, five days each week, for 50 weeks each year. To this figure one should add a reasonable level of maintenance and support which is particular to graphics such as, for example, $5 to $10 per hour per console. (Obviously this figure would not rise proportionally to the number of consoles; but an estimate for maintenance and support is needed for the number of consoles that may be projected.) These latter estimates increase the hourly rates to a range from $12.50 to $30. This resultant can, of course, be easily modified contingent upon other assumptions for utilization, amortization schedule, maintenance and support, and other particular hardware/software options. It is now convenient to consider the wage rate (including fringe benefits, overhead, etc.) for the principal ICG user. This is another parameter. As an example, assume that it is $20/hour. Then the increased cost of $12.50 to $30 per hour associated with the provision of ICG services can be ratioed with respect to wage rate to produce a figure of "productivity increase" necessary to defray costs of the ICG system. That is $12.50/$20 through $30/$20 or 0.625 through 1.5 increase in productivity. Put another way, the graphic console user would have to produce 1.625 to 2.5 his normal output (productivity ratio) to defray additional system costs. This range depends on factors described earlier but are put forth as "reasonable" figures of merit to delineate the principal cost components. It is important to note at this juncture that trends toward lower costs of hardware and software and higher costs of manpower lead to a decreased range of productivity ratios which are necessary to defray ICG systems cost. However, even present productivity requirements favor the widespread use of ICG. This is because experience has indicated, in somewhat qualitative terms, that most applications have shown a minimum productivity ratio of three. Therefore, an application with sustained utilization should show a cost benefit based on the statistics set forth here. Again, one should be cautioned that such statistics and judgments might well have detractors. The experiences and judgments of practitioners of ICG vary widely as might be expected. There is no doubt that the amount of hard data on the subject is quite inadequate. Somewhat more potential exists than has been exploited to evaluate the expected cost benefits in employing ICG for a particular application.

TECHNIQUES FOR DETERMINING COST REDUCTION

The preceding discussion has dealt primarily with background principles which impact cost benefits determinations. The sequel will describe more specific techniques for determining cost reduction and the number of graphic consoles that will support multiple graphic applications.

Any application under consideration might be classified into cost components. For example, suppose T_1 represents the total cost to accomplish a particular application using current technology. Suppose further that T_1 is partitioned into costs as follows:

A_1 = Planning and Scheduling

B_1 = Set Up

C_1 = Key Punch and Data Preparation

D_1 = Computing

E_1 = Data Analysis and Evaluation

F_1 = Documentation and Administration

G_1 = Correlated functions unaffected by the introduction of ICG.

Here A_1 through F_1 represent component tasks whose costs will be affected (either increased or decreased) by the introduction of ICG. The element, G_1, represents components unaffected by ICG. Similarly T_2 is composed of the same components A_2, B_2, ..., G_2 after ICG has been introduced. Of course, one may choose whatever classification components that fit the application. If desired, classification can be somewhat more detailed. For the present example, the two costs, T_1 and T_2, might be represented by two bars as follows:

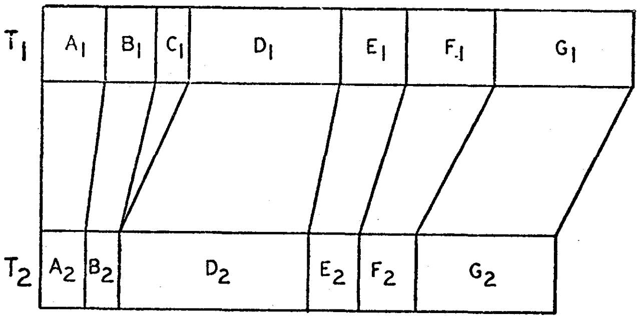


Figure 1—Component costs for application without and with graphics

In the example, keypunching is eliminated, computing costs are actually increased, and other costs (except for G) are decreased. Then,

$$\text{Cost Reduction} = \Delta T = T_1 - T_2 = (A_1 - A_2) + (B_1 - B_2) + (C_1 - C_2) + (D_1 - D_2) + (E_1 - E_2) + (F_1 - F_2)$$

($G_1 - G_2 = 0$ since the G's are defined as those tasks unaffected by ICG.)

It should be noted that the two primary parameters that influence the value of ΔT are man hour rate of the ICG console user and the cost per hour rate for the use of the ICG system. Thus,

$$\Delta T = F(R_M, R_C)$$

where R_M = Man Hour Rate

and R_C = Console Rate

Both T_1 and T_2 and hence ΔT will be affected by the choice of R_M and R_C. More specifically, ΔT may be expressed as:

$$\Delta T = K + R_M\ (H_1 - H_2) - R_M H_3 - R_C H_3$$

where: K = The cumulative cost effects (gains and/or losses) resulting from the use of ICG but not directly a function of the graphic system or of the graphic console user, per se.

H_1 = The hours the potential graphics user would employ in the conventional task where cost is defined earlier as T_1.

H_2 = The hours (expected to be less than H_1, in general) that the user would expend after the interjection of ICG, but not in the use of the graphic console, per se.

H_3 = The hours that the ICG user spends at the console.

The parameter, K, could include such effects as material or weight reduction, decreased inspection requirements, decreased batch or other nongraphic computation, different technician and administrative support, and other factors with varying degrees of direct cost effects. Since K would normally be expected to be positive, on balance, the required console user productivity gain needed to defray ICG systems costs would be decreased. If it is assumed that R_M and R_C are measured in \$/hour, then H_1, H_2, and H_3 are measured in hours.

The preceding equation for ΔT encompasses all gains and losses that result from ICG use. This equation is rewritten as:

$$\Delta T = K + (H_1 - H_2 - H_3) R_M - H_3 R_C \qquad (1)$$

Here it is clear that ΔT is a linear function of R_C and that for any R_M, the slope of the line is $-H_3$. For an application under consideration to use some form of ICG, ΔT can be plotted as in Figure 2.

Using algebra on equation (1), it can be shown that

$$\Delta R_C = (H_1 - H_2 - H_3) \Delta R_M / H_3 \qquad (2)$$

Where the line, for a particular R_M, crosses the abscissa, that will be the highest console rate (threshold) to achieve cost effectiveness. Either the plot for ΔT or equation (2) may

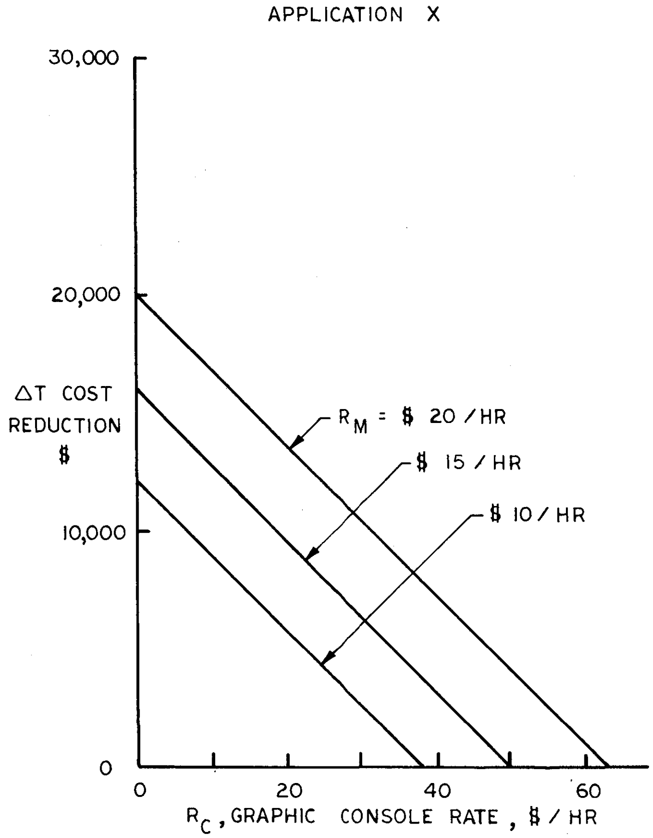


Figure 2—Example application cost reduction versus console rate for increments of manpower rate

be used to relate the change in the threshold graphic console rate to alternative man hour rates.

The y-intercepts of equation (1) show the maximum cost reduction, ΔT_{MAX}, that would be achieved for a given man hour rate. This would be the value of ΔT for a zero console rate. The formula for ΔT_{MAX} is:

$$\Delta T_{MAX} = K + (H_1 - H_2 - H_3) R_M$$

It should be noted that ICG becomes increasingly cost effective as graphic rates decrease and as man hour rates increase. Thus, present economic trends of lower computer costs and higher wages trend toward the increased value of ICG. The higher ΔT for higher R_M at a particular R_C means that the introduction of a labor saving system such as ICG is worth more when it increases the productivity of high-priced labor. It does *not* mean that tasks performed by low-cost labor should now be performed by high-cost labor. In the example, the threshold for cost reduction for a $20/hour manpower rate is about $62/console hour. In other words, ICG can benefit application X, in pure cost, for a total graphic rate as high as $62/hour which would imply that application X can support a fairly sophisticated ICG system. The $10/hr. R_M crosses the axis at a console rate of about $37/hr. This means that if application X uses principal manpower at a rate of $10/hr, then the ICG system would have to be somewhat less expensive if it is to be a paying proposition in purely monetary terms.

EXAMPLE OF GRAPHIC/NON-GRAPHIC COMPARISON

To illustrate the preceding discussion of cost reduction, a candidate application in the Lockheed-Georgia CADAM evaluation for potential contract work will be described.

The following chart is a breakdown of tasks for both conventional and CADAM processing of a particular numerical control tape development task, Figure 3.

From the chart, it can be observed that tasks 1, 4, 5, 10, and 13 are unaltered. They would fall into the classification, "G," in the earlier discussion of cost reduction. All other tasks are altered because of graphics and would be treated similar to classifications "A" through "F".

In reference to the equation for ΔT in terms of K, H_1, H_2, H_3, R_M and R_C, it is noted for this example that K is zero (although there are certainly other related effects, like inspection, that are not included in the chart).

The formula, $\Delta T = K + (H_1 - H_2 - H_3) R_M - H_3 R_C$

gives $\Delta T = 0 + (40.55 - 6.10 - 9.26)\ 20 - 9.26\ (40)$

Based on a 4-scope CADAM system with a $40/hr console rate, R_C, and a $20/hr manpower rate, R_M.

Thus $\Delta T = \$133.40$ (cost reduction)

Since tasks 1, 4, 10, and 13 are unaffected by graphics (classification "G"), they need not necessarily have been included in the chart. Their inclusion does help set forth all the elements of the example N/C application. If they had been deleted from the chart, then both H_1 and H_2 would be 6.10 hours less. That is, they would be 34.45 and 0 respectively. As discussed before, this would not alter the cost reduction value of $133.40 because cost reduction is independent of tasks classified as "G."

CONSOLE RATES

The applications and system costs of a projected Lockheed-Georgia CADAM system relate directly to the earlier discussion on the optimum charging mechanism and on the opti-

	CADAM		CONVENTIONAL		COMMENTS
	H_2 Hours	H_3	Hours, H_1		
TASKS	Off-Scope	Scope	Preparation	Program	
1. Collect and analyze data, drawings, fixture design, etc.	1.00		1.00		
2. Machine part drawing for layout		2.37	6.00		
3. Furnish tool dept. with fixture sketch		.50	2.00		
4. Determine cutters	.30		.30		
5. Plan cut sequences	.50		.50		
6. Generate cutter paths		6.39		24.00	Conventional method is by written manuscript
7. Punch IBM cards			.30		Includes all later card punch corrections
8. Assemble card deck			.05		Includes changes to deck
9. Submit cards to computer			.10		Includes all computer runs
10. Check center-line and postprocessor data from computer listing	.30		.30		
11. Plot N/C Machine tape			1.50		CADAM eliminates this, usually
12. Check N/C plot			.50		Overlay on N/C layout or visual check
13. Draw operation sequence sheets	4.00		4.00		
TOTALS	6.10	9.26	16.55	24.00	

Figure 3—Example numerical control task

APPROXIMATE IBM 360/65 SYSTEM COSTS (IN THOUSANDS OF DOLLARS)

Number of Scopes	2	3	4	5	6	7	8	10	11	12	13
Hardware Rental/Mo.:											
CPU & Associated Peripherals	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0
Scopes (Includes 10% 2nd Shift Charge)	4.2	6.3	8.4	10.5	12.6	14.7	16.8	18.9	21.0	23.1	25.2
512 Bytes Mass Storage	3.6	3.6	3.6	3.6	3.6	3.6	3.6	3.6	3.6	3.6	3.6
Scope Controller	5.1	5.1	5.1	10.1	10.1	10.1	15.2	15.2	15.2	15.2	20.2
Display MPX			0.5	0.5	0.5	0.9	0.9	0.9	0.9	1.4	1.4
Printer Switch; 6 Disc Packs	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
Drum, Controller, Selector Channel, MPX Sel. Sub Channel				7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5
Total Hardware Rental	23.5	25.6	28.2	42.8	44.9	47.4	54.6	56.7	58.8	61.4	68.5
Programming, Operator Support	8.6	8.6	8.6	8.6	8.6	8.6	8.6	8.6	8.6	8.6	8.6
Total Monthly Cost	32.1	34.2	36.8	51.4	53.5	56.0	63.2	65.3	67.4	70.0	77.1
Hours of Use/Month, (2 Shifts, 5 Days/Wk., 80% Utilization)	554	831	1108	1385	1662	1939	2216	2493	2770	3047	3324
Cost, $/Scope Hour	58	41	33.5	37	32	29	28.5	26	24	23	23
Cost Adjustment/Scope for Errors in Estimates, Assumptions, Unanticipated Costs	10	8.0	6.5	5.5	5	4.5	4.0	3.5	3.0	2.5	2.2
Total Cost, $/Scope Hour	68	59	40	42.5	37	33.5	32.5	29.5	27	25.5	25.2

Figure 4—Console rate versus number of scopes

mum number of scopes to support the applications. Data are being compiled within the framework of an on-site CADAM system. It is assumed that a dedicated IBM 360/65 configuration will be acquired to service the appropriate number of scopes. Based on the potential of driving "N" scopes two shifts each day, five days each week and with 80 percent utilization, the hourly console rate as a function of "N" has been computed, as shown in the preceding tabulation, Figure 4. The hourly console rate for this particular system is high for up to three consoles (in comparison to ranges set forth earlier) because of the cost of the assumed graphic dedicated IBM 360/65 which is predicated on multiple-scope use, because third shift CPU operation is not assumed, and because substantial support is to be given initially.

PRODUCTIVITY RATIO DEFINITION

One of the more valuable measures of effectiveness for graphics is productivity ratio defined by:

$$\text{Productivity Ratio} = \frac{R_M + R_C}{R_M}$$

For a R_M of 20 and a R_C of 40, as in the example above, this ratio would be 3. In other words, one hour at the console would have to produce the equivalent of three hours of non-graphic work to defray the extra cost of the console. From the chart of Figure 3, the conventional hours on which graphics have an effect are 40.55-6.10=34.45. This work can be accomplished, according to the chart, with 9.26 hours at the console. Thus (34.45/9.26) = 3.7 is the productivity ratio for the example. This compares favorably to the 3.0 threshold or "breakeven" ratio. In computing ratios of hours as exemplified here, it is essential to consider only graphic-affected tasks if the ratio is to have meaning. That is why 6.10 was excluded from the calculation of the ratio of 3.7. That is, 6.10 was subtracted from both the total CADAM hours and the conventional hours. Had this not been done, the ratio would have been (40.55/15.36) = 2.6 which would have falsely indicated that the increased productivity did not reach the "breakeven" value of 3.0. Furthermore, had the unaffected hours been much greater, say for example 106.10 instead of 6.10, the ratio would have been (140.55/115.36) = 1.2. This illustrates that if the constant hours (same for both conventional and graphics) are not subtracted out before calculating

Task	Conventional Hours	Cumulative Conv. Hours	Scope Hours	Cumulative Scope Hrs.	Productivity Ratio *	Cum. Prod. Ratio	# Scopes (2770 Hours/Scope)
1. Stock Drawings	632	632	146	146	4.3	4.3	1
2. Assemblies, Detail Drawings	18940	19572	5167	5313	3.7	3.7	2
3. Sheet Metal Drawings	7975	27547	2175	7488	3.7	3.7	3$^-$
4. Extrusion Drawings	1873	29420	577	8065	3.2	3.6	3
5. N/C (Tapes)	1780	31200	668	8733	2.7	3.6	3$^+$
6. Detail Drawings (Another Project)	5502	36702	2267	11000	2.4	3.3	4
7. *Other Tasks: Layouts, Tool Design, Etc. (Estimated Ratios)					2.2	3.1	5
					2.0	2.9	6
					1.9	2.75	7
					1.8	2.6	8
					1.8	2.5	9
					1.7	2.4	10
					1.7	2.35	11
					1.7	2.3	12

* See following discussion in text.

Figure 5—Average productivity ratio for applications using N scopes

the ratio, then the ratio would be a function of the number of unaffected hours which are listed and would, therefore, obviate the meaning of the ratio as a measure of effectiveness.

The console rates from Figure 4 are used in Equation 3 (with $R_M = 20$) to determine productivity ratios for each "N."

These productivity ratios, as a function of N, are the "breakeven" ratios required for "N" scopes. The plot of this ratio as a function of "N" is depicted as function A on the graph, Figure 6.

APPLICATIONS AND PRODUCTIVITY RATIOS FOR CADAM

The current Lockheed-Georgia mix of potential applications for CADAM has been initially identified as follows:

Engineering Drawings

- Layouts
- Assemblies, Details
- Machined Parts
- Sheet Metal Parts
- Extrusions
- Stock Drawings

Tool Design

- Interference Layouts
- Small Fab and Assembly Tools

N/C (Tapes)

- Lay-up Blocks
- Production Machined Parts

For several specific tasks, the productivity ratio has been computed based on studies, experimentation, and judgment. The ratios are believed to have been conservatively established. It is assumed that the first scopes will be applied to the most productive of the potential application tasks. Therefore, as a larger number of scopes (N) are contemplated, applications with less productivity are brought into consideration and thus the cumulative productivity ratio will decline as a function of "N." These potential applications are listed in order of decreasing productivity ratio in the preceding tabulation, Figure 5.

It should be noted that the application set beyond four scope utilization has not been determined as of this writing. Therefore, for the purpose of showing the complete technique for ascertaining the "optimum" number of scopes, estimates of ratios are inserted on the basis of an average ratio for complete utilization of each additional scope. Subsequent studies may very well show additional applications with relatively high ratios. However, at any assessment time, the applications should be added to the set in descending order of ratios—as exemplified in the tabulation.

The CADAM operational use was studied for a projected 10-month period, from initiation on March 1, 1975, through the end of 1975. Therefore, the hours of use to determine the number of scopes (last column of Figure 5) is obtained by multiplying the monthly use (lower part of the tabulation on Figure 4) by 10.

ANALYSIS OF GRAPH OF PRODUCTIVITY RATIOS

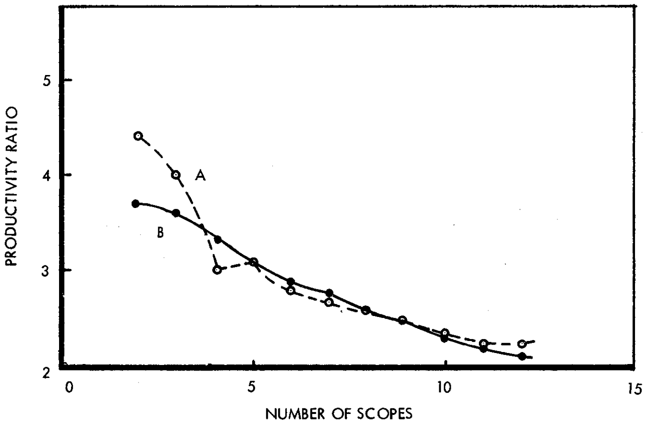
Curve B, Figure 6, of productivity ratio versus "N" results from plotting the data from the table of Figure 5. It is interesting to note that both functions A and B decline, in general, with increasing "N." Function A is erratic because of the introduction of certain specific hardware, like controllers, drums, etc., at specific values of "N." Thus, function A is more or less a step function. Similarly, function B depends on the mix of application ratios which is not homogeneous in this case. Therefore, function B is erratic.

If function B remained below function A for all "N," then there would be no number of scopes that would make CADAM cost effective for the considered applications.

The crossover between "N" equals three and four, illustrates the hypothetical point where savings (increase in productivity) exactly defray the cost of the scopes. The differential productivities between "B" and "A" is a maximum for "N" equal to four. This means that the increase in productivity obtained by going to four scopes more than offsets the additional systems costs. From then on, increasing "N" yields a decreasing difference. Where the functions cross again, between 8 and 9, all cost benefits are wiped out. Thus, the optimum for the particular application mix considered and for the assumed configuration costs is four scopes. Naturally, any shift in either the "A" or "B" functions would, in general, change the optimum number of scopes in the purely cost benefits sense.

POSITIVE AND NEGATIVE INTANGIBLES

The data that is used to develop function B is somewhat more subjective than that used for function A. Although the methodology for evaluating cost effectiveness, as set forth here, is an attempt to blend the maximum amount of both objective and experienced subjective data, there must remain some level of error—both positive and negative error. To those intangible benefits cited in the opening remarks should be added the ability to expand advanced technology into new areas, better manpower leveling and manpower utilization, better interfaces and understanding between specialty disciplines, and better interfaces and understanding between major functions such as Engineering and Manufacturing. These qualities improve the opportunities for better planning and scheduling which will bring about significant cost benefits. On the negative side, it is difficult to predict what special systems problems will arise, special problems in scheduling two-shift operation may arise, contract go-ahead may not be compatible with system availability, it may be difficult to schedule the most productive mix of applications at any given time, estimates may not have anticipated all relevant manpower factors, etc. On balance, however, it is felt that the estimates set forth in this paper are conservative.



A - BREAKEVEN RATIO (REQUIRED TO JUST PAY FOR SYSTEM)
$ PER HR COST =(RATIO X $20)- $20
E.G. 4 SCOPES, RATIO = 3, COST = $40/HR

B - ESTIMATED CUMULATIVE PRODUCTIVITY
RATIO (ENCOMPASSES MORE AND MORE
TASKS WITH LOWER PRODUCTIVITY RATIOS)

Figure 6—IBM 360/65 graphic system dedicated to CADAM

It is not common practice for most organizations to know the breakdown of costs for conventional tasks not to mention the breakdown for tasks using advanced technology. This is not to say that such data could not or should not be sought. On the contrary, in an era where increased productivity is becoming so very important, it would seem that organizational growth or even survival might well depend on the intelligent appraisal of present costs of operation and of alternative costs as a function of the incorporation of new technology.

As pointed out earlier, although cost benefits are by no means the only benefits, they undoubtedly play a major role in the decision processes of most establishments. Even with this constraint, interactive computer graphics is now coming of age.

ACKNOWLEDGMENT

A paper of this kind is, of course, the product of inputs from many sources. In particular, I would like to thank Ira Cotton of the National Bureau of Standards and Don Scarbrough, head of the Lockheed-Georgia Operations Research Department, for a very constructive critique; Dick Schunk, Staff to the Lockheed-Georgia Chief Design Engineer, for furnishing excellent background and operational experience on the local CADAM evaluation; Roy Harris and the Lockheed-Georgia Scientific Computing organization for CADAM systems costs data; Harry Peabody of Lockheed-Sunnyvale for experience

data on CADAM; and many people from Lockheed-Burbank, developers of CADAM

BIBLIOGRAPHY

1. Chasen, S. H., *"General Principles for Planning an Interactive Graphic System,"* UCLA Short Course (Lecture Notes), Engineering 819.53, *Applied Interactive Computer Graphics*, 23-27 September 1974.
2. Cislo, R. A., "Graphic System Performance Evaluation," (*PROC*) *1972 ACM National Conference.*
3. Cotton, I. W., *Cost Benefit Analysis of Computer Graphics Systems*, National Bureau of Standard Technical Note 826; April 1974.
4. Foley, J. D., "An Approach to the Optimum Design of Computer Graphics Systems," *Communications of the ACM*, June 1971.
5. Jacobs, L. D., *CRT Graphics Consoles—An Aid to Selection*, Rome Air Development Center, Technical Report 71-61, Nov. 1971.
6. Machover, C., "How Applications Affect Interactive CRT Terminal Selection," UCLA Short Course (Lecture Notes), Engineering 819.53; *Applied Interactive Computer Graphics*, 23-27 September, 1974.
7. Poole, H. H., "Computer Display System Tradeoffs," in *Computer Graphics: Techniques and Applications*, Eds. Parslow, Prowse & Green; New York, Plenum Press, 1969.

Area Director:
Earl C. Joseph
Sperry Univac Defense Systems
St. Paul, Minnesota

Future prospects in data processing

So far, in the seventies, rapid revolutionary developments have occurred in the development and usage of computer systems. Already, we have passed through three generations of microprocessors, the distributed-function architecture (networks) has emerged, dedicated special-purpose systems have reversed the economy-of-scale rule, and operating systems are turning toward easier to use systems. Massive evidence now exists, pointing toward a new computer era for the near and far term futures. These sessions will forecast trends and discuss future prospects relative to the direction that these trends are pushing the computer field especially relative to expected impacts of interest to users. The three sessions will cover the following:

Session I—Future Maxi-Micro Technology, Economics and Operations.
Session II—Future Distributed Function Networks Technology and Management.
Session III—Panel—Expected Future User Impacts.

Where is technology taking us in data processing systems?

by BERNARD J. GREENBLOTT and MU Y. HSIAO
International Business Machines Corporation
Poughkeepsie, New York

INTRODUCTION

During the past decades progress in LSI for logic and storage implies the following for data processing systems: (i) Cost reduction for a given function; (ii) Complexity increasing which also implies Capability and Performance increase; (iii) Reliability increase; (iv) Mass production capability handled by automated processes; and (v) Reduction in physical size. Similarly, in magnetic recording technological progress toward an increase in areal density of storage has the following implications: (i) Cost reduction per bit stored; (ii) Mass on-line storage is possible and practical; (iii) Performance improvement; and (iv) Reduction in physical size.

As a result of this technological progress in LSI, high density magnetic recording, and related device development have opened up three broad areas of significance to data processing systems:

(1) Hardware costs of storing and processing data are decreasing, i.e., the unit cost of functional work performed by a computer is decreasing.
(2) Following the trend described above, data processing systems of the future will be designed around the user, not only in terms of business and domestic requirements, but also taking into account the physical location of the computer and/or the user's terminal.[1,2]
(3) Advances in device technology have paved the way for overall system improvement.

Based on existing trends in technology as outlined above, we can visualize a data processing system of the future according to the concept shown in Figure 1.

With reference to Figure 1 a self-contained unit with intelligent terminals or other I/O devices at the user's end is essentially the whole on-line data base system.

As developed in the following sections of this paper the trends of technology over the last decades have been moving toward this concept. What is projected from observed trends is therefore no more than the final convergence of these trends.

THE TRENDS OF TECHNOLOGY

Progress in semiconductor devices for logic and storage

Judging from recent trends/extrapolations, IC's with 1 million components, or 10 bipolar LSI chips to yield 100,000 gates for computer application are possible for the 1980's.[3]

These achievements, representing a definite trend from 1953 to 1974, (Figure 2) show an increase in computer processing capability and a decrease in physical size, quantified as follows: (1) From milliseconds (10^{-3}) to nanoseconds (10^{-9}) in circuit performance means 6 orders of magnitude; (2) From cubic inches per circuit to 10 thousands of circuits per cubic inch means 6 orders of magnitude; and (3) At least 3 orders of magnitude improvement in power requirement and circuit failure rate.

Storage technology[4,5]

In reviewing different conventional storage technologies, E. W. Pugh[5] gives the following views on cost reduction as shown in Figure 3.

Where there is a correlation between the cost/performance and the size of the largest single unit in that system, it is predictable that trends of the future will be to achieve ratio improvements of 40 or even greater in the cost reduction and volumetric efficiency of storage systems.

Technological trends of various storage devices are discussed as follows.

Semiconductor storage

Semiconductor storage offers the best example of cost reduction, performance and volumetric efficiency improvement based on LSI technological progress.

Progress in semiconductor storage has been made with both bipolar and FET devices (see Figure 4). Semiconductor storage at the present has a role in supporting innovative storage arrangements/hierarchies (cache, buffer, local store). Semiconductor storage, in combination with magnetic storage, yields increased volumetric efficiency and cost performance improvement.

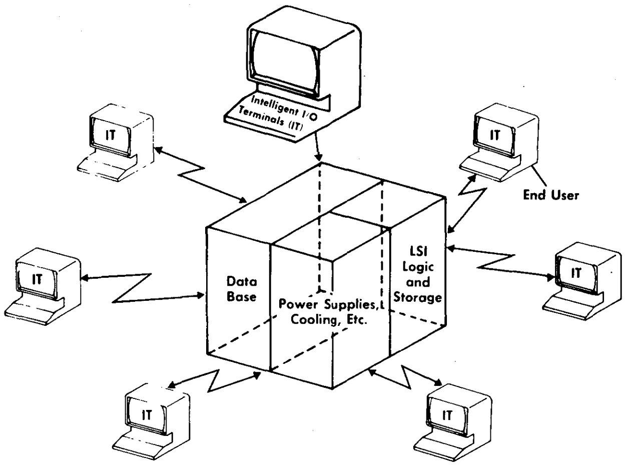


Figure 1—Trends in future processing systems

It is likely that developments in storage systems over the next several years will also emphasize the organization of the storage, and the information in storage, as well as the technology of the storage devices.

Magnetic storage[6-8]

In the current state-of-the-art, disks, drums, and tapes are usually considered as the major source for mass storage because of their large capacity, and low cost. Technology progress in recent years has included not only read/write heads, but media, source coding and error correction, read/write channel design and mechanical systems, resulting in cost-per-bit decrease, and improvement in performance and reliability. Areal density has been used as a key factor in achieving cost reduction and improved performance in magnetic recording devices.

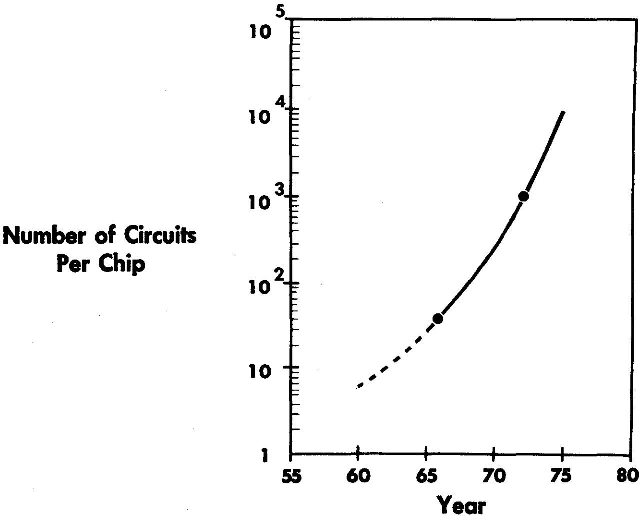


Figure 2—Logic circuit density progress

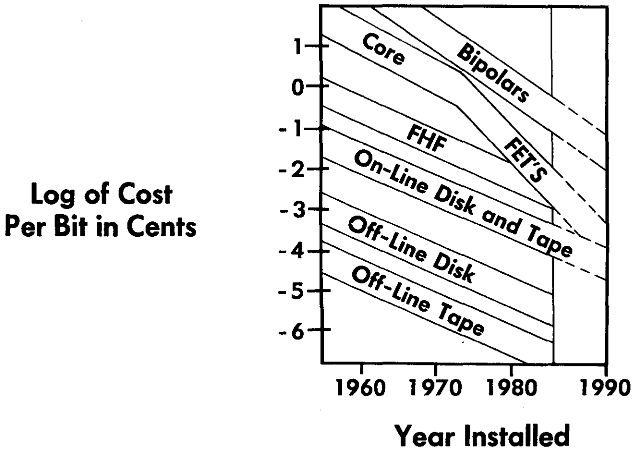


Figure 3—Storage technologies

Figures 5 and 6 show the progress in areal density for disk and tape products.

Progress in terminals

Since the late 1960's, applications of terminals have been rising rapidly. An indication of this trend is reported in the Infotech State-of-the-Art Report on Terminals,* which predicts the number of data terminals in the U.S. to grow from 195,000 in 1970 to 2,425,000 by 1980. This estimate may be conservative, however, in view of what has been observed in the electronic calculator market during the past few years. Clearly, LSI technology is exercising a

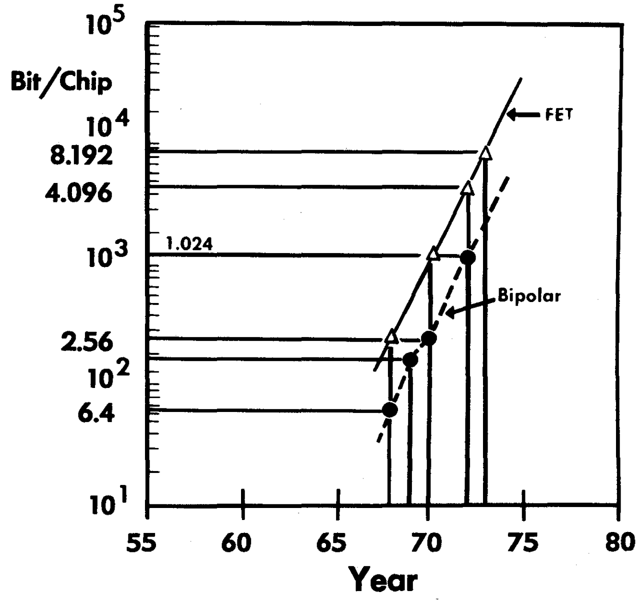


Figure 4—Progress in semiconductor storage

* Computing Terminals, International Computer State-of-the-Art Report, Maiden Head, England, Infotech 1971, #4

major thrust for terminal progress. LSI, which puts more and more function into smaller elements, has made it possible to incorporate more storage and logic capabilities into the terminals.

As a result of technological progress it is expected that the new terminal-based system together with the rapid-growth of data-base and communications-based systems will put more capability in the terminal subsystems.

Multi-level hierarchies in the structure of the distributed system are becoming more likely, thereby reducing the burden on communication lines, and increasing system availability.

A WORKING EXAMPLE

In this section, an attempt is made to project the end result of the technological progress described previously.

Since the principal result of technological progress has been to reduce the cost of processor and storage functions, as well as the physical size of a data processing system, let us therefore visualize the single shippable unit, as shown in Figure 7.

Based on projections derived from trends observed over the past decade the configuration shown in Figure 7 consists essentially of 2 technologies:

1. *LSI Technology*—Several LSI processors that yield data processing power rated in MIPS, and an LSI storage of 10^9 bits.
2. *Magnetic Recording Technology*[9]—An archival store at 10^{12} bits is based on the assumption that 10^7 bit/in^2 areal density of magnetic storage.

The hardware in Figure 7 can be represented by the system block diagram in Figure 8.

To quantify the data processing system shown in Figure 7, let us assume the access time of the storage system, and then work back to the required computing power of the processors. A set of assumptions for computation purpose on the characteristics of the storage system is given as follows: (1) Total storage size $=10^{12}$ bits; (2) Average access time/data block (e.g., 512 bytes) $=5\mu s$ at the entrance; (3) The hit ratio (Hr) of the cache is 0.985; (4) $E/B=.1$ (Ratio of instruction per byte of required data) of processors; and (5) The block size transferred between the processor and cache is 8 bytes.

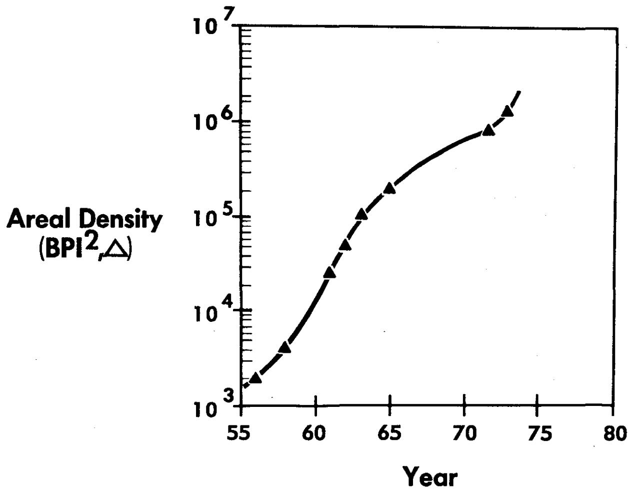


Figure 5—Increases in areal density in disk products

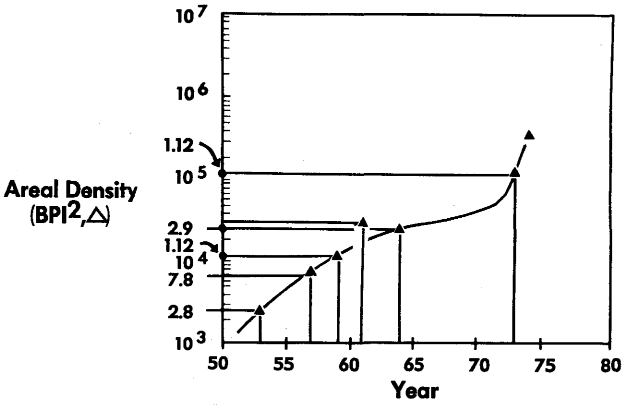


Figure 6—Increases in tape areal density

Using these assumptions, the required computing power of the processor can be calculated from Figure 8. The activity (=data traffic) between the processor and the cache is equal to

$$(F_1+S_1)B_1 \tag{1}$$

where B_1 is assumed to be 8 bytes. The activity between the cache and the storage system is equal to

$$(F_2+S_2)B_2=(F_1+S_1)(1-H_r)B_2 \tag{2}$$

where B_2 is assumed to be 512 bytes. Therefore, we have

$$\frac{(F_1+S_1)B_1}{(F_2+S_2)B_2}=\frac{(F_1+S_1)B_1}{(F_1+S_1)(1-H_r)B_2}=\frac{8}{.015\times512}\cong1 \tag{3}$$

This shows that the data activity between the processor and the cache is equal to the rate between the cache and

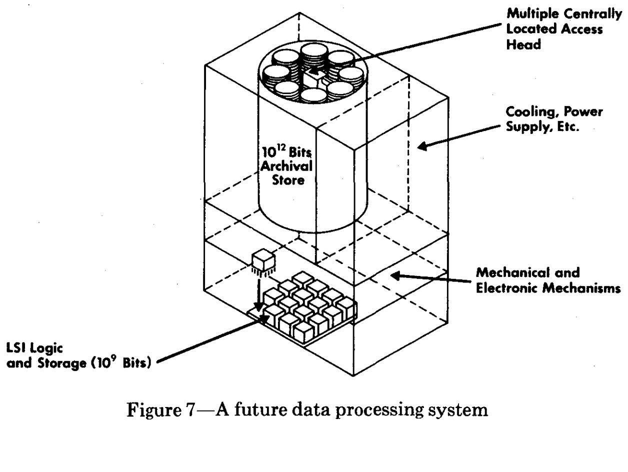

Multiple Centrally Located Access Head
Cooling, Power Supply, Etc.
10¹² Bits Archival Store
Mechanical and Electronic Mechanisms
LSI Logic and Storage (10⁹ Bits)

Figure 7—A future data processing system

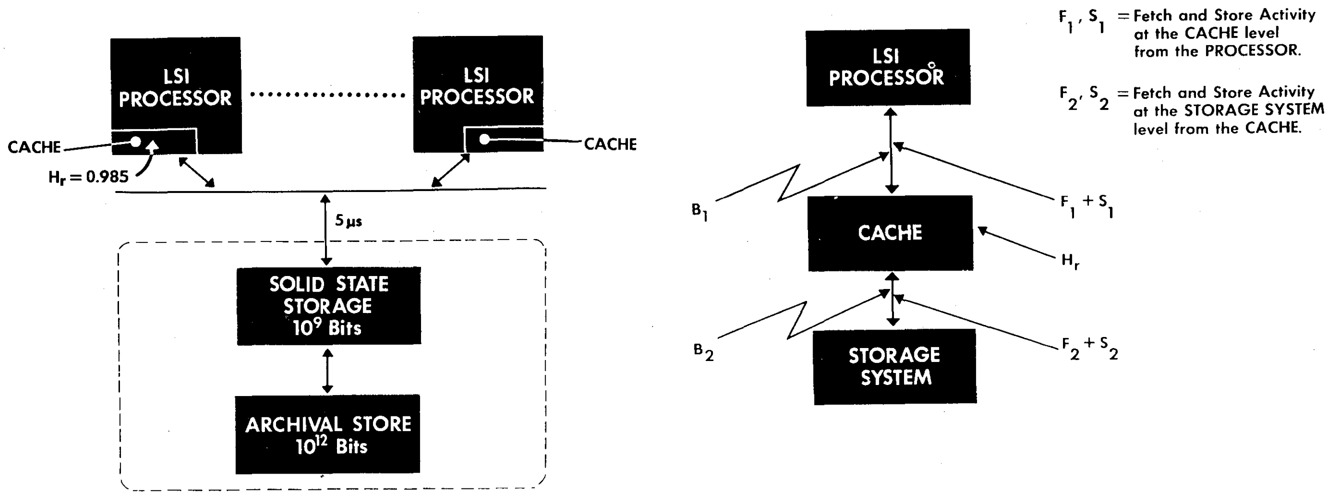

LSI PROCESSOR
LSI PROCESSOR
CACHE
CACHE
$H_r = 0.985$
5µs
SOLID STATE STORAGE 10^9 Bits
ARCHIVAL STORE 10^{12} Bits
LSI PROCESSOR
CACHE
STORAGE SYSTEM
B_1
B_2
$F_1 + S_1$
H_r
$F_2 + S_2$
F_1, S_1 = Fetch and Store Activity at the CACHE level from the PROCESSOR.
F_2, S_2 = Fetch and Store Activity at the STORAGE SYSTEM level from the CACHE.

Figure 8—System block diagram

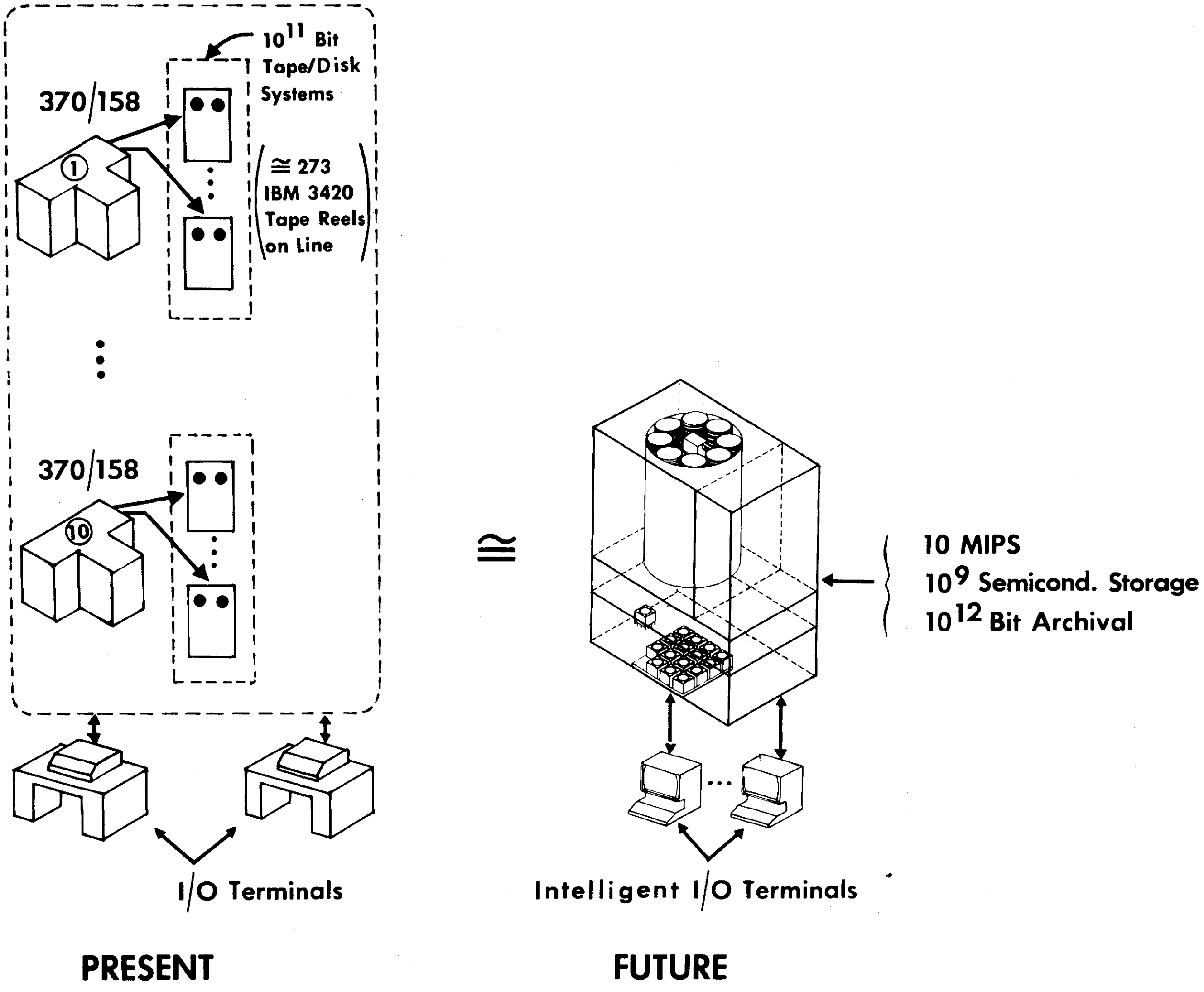


Figure 9—Progress in a data processing system

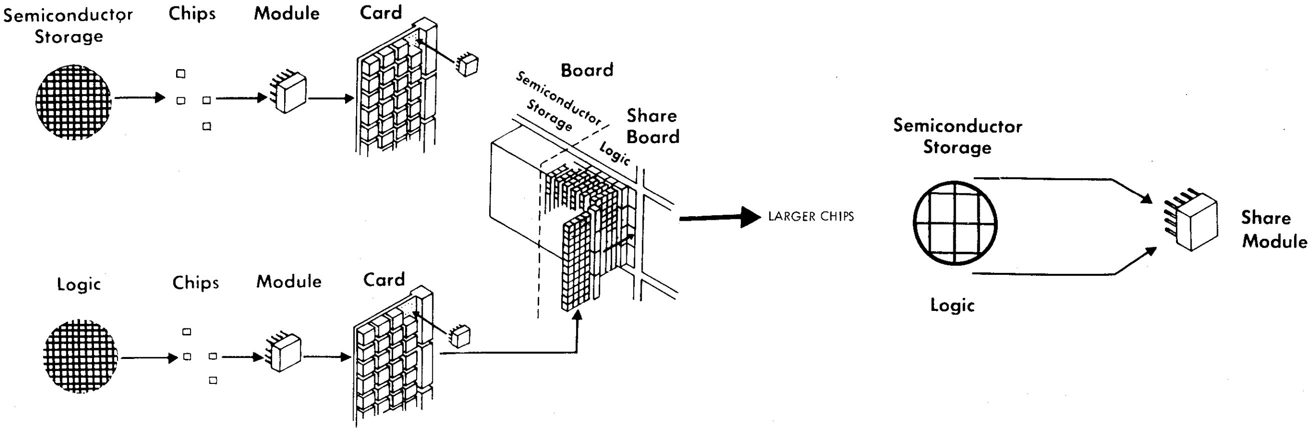


Figure 10—Chip packaging/merging concept

the storage system, which equals

$$\frac{512 \text{ Bytes}}{5\times10^{-6} \text{ sec}} \cong 100 \text{ MB/sec} \qquad (4)$$

Based on the assumption that each instruction of the processor on the average requires 10 bytes of data, i.e., $E/B=0.1$, a 100 MB/sec data rate can be supported by a processing power of 10 MIPS. Therefore, as predicted by known progress in LSI technology, a 10 MIPS processing power can be hosted within several LSI processor modules as shown in Figure 7.

As a final comparison, let us consider how this system compares with the IBM 370/158 where machine characteristics can be generalized as follows: (i) Processing capability=1 MIPS; and (ii) Storage capacity=4 MEGA Bytes.

If we effectively put ten IBM S370/158's together to yield system characteristics of 10 MIPS processing power and $10\times(4\times8\times10^6 \text{ bit})=3.2\times10^8$ bit storage, we can say that the ten systems together with more than 2000 reels of tape on-line are about equivalent to that of the data processing system as shown in Figure 9.

Figure 9 shows the natural trend of future data processing systems, and the need to design these systems so as to share the merits of technological progress in different areas.

FUTURE TRENDS

Advances in device technology have made a great impact on computer systems, affecting not only cost, performance, and reliability but also volumetric efficiency.

In the logic area, it is possible to fabricate hundreds and even thousands of logic gates on one chip. In the semiconductor storage area, we have witnessed the progress of 64, 128, up to 8192 bits per chip.[10] This trend will continue in the future and it will mean further reduction in the cost per bit of solid state storage. The realization of these cost reductions will also require innovations in manufacturing procedures. If the technology is the same, then a merge of logic and storage in the LSI process is more desirable. This means that the chip size will also increase. The result of the trend toward the large chip is a reduction in packaging/handling costs. Figure 10 shows the trend of merging of logic and storage on the same board, eventually to the same module.

Improvements in LSI technology common to storage and logic have established that storage/logic functions will no longer be distinguishable in any physical way. Furthermore, development in the storage hierarchy concept is the key link between high speed LSI semiconductor storage, and mass storage magnetic recording devices.

In magnetic recording devices, the trend in areal density is much the same in terms of factoral improvements in cost and size reduction. We see disk and tape becoming inseparable, with an access time of a fraction of a second. Mass on-line data storage is essentially the archival store with its storage capacity being the total system store capacity.

CONCLUSION

During the past decade technological advances in semiconductor logic and storage, and increased areal density in magnetic recording—particularly tape and disk—have been achieved. Initially the benefits of these achievements were applied to desk-top calculators, small computers, and aero-space equipment. However, as a result of engineering innovation and the solution of manufacturing and process problems, it is becoming apparent that these technological advances will also have an effect on the concept and performance capabilities and applications of terminals in front of large scale data processing systems of the future.

What we have emphasized here is that the new concept is principally the result of a convergence of many technological trend-curves observed over the past decade, i.e., the focus of attention is not on any breakthrough technology but, rather, is on the net result of several overlapping/evolving technologies and where they are taking us in data processing systems. The significance of a 10^{12} data base storage with 10 MIPS processor power packaged in a single unit is thus completely overshadowed by a corollary effect of a cost-per-bit to the user which is expected will eventually be somewhere in the range of the cost of telephone service.

No longer will data processing capabilities be cost-prohibitive to many potential small users. Decreased terminal and system cost, and decreased functional cost to the user, will make computer services available to the needs of the local community, of small businesses, and the general public-at-large. In the context of access to the system via an interactive terminal the potential for new aids to education, problem-solving, and decision-making is almost limitless.

ACKNOWLEDGMENT

The authors express their appreciation to Mr. J. A. Haddad for his encouragement in developing this paper.

REFERENCES

1. Tobias, M. J. and G. M. Booth, "The Future of Remote Information Processing Systems," *FJCC 1972*, pp. 1025-1035.
2. IEEE Computer Society, *"Computer"—Distributed-Function Computer Architectures*, March 1974.
3. Heath, F. G., "Large-Scale Integration in Electronics," *Scientific American*, February, 1970, p. 31.
4. Feth, G. C., "Memories are Bigger, Faster—and Cheaper," *IEEE Spectrum*, November, 1973.
5. Pugh, E. W., "Alternatives to Magnetic Recording," *AIP Conference Proceedings*, #5, pp. 31-44, 1971.
6. Harker, J. M. and H. Chang, "Magnetic Disks for Bulk Storage, Past and Future," *SJCC*, 1972.
7. Branscomb, L. M., "Technology Trends in Peripheral Devices," *IEEE 1974 Compcon Spring*, pp. 17-20.
8. Conley, S. T., "Tape, Disk . . . What Next?" *IEEE 1974 Compon Spring*, pp. 5-8.
9. Pohm, A. V. and R. J. Zingg, "Proposal for a 10^{12} Bit Flexible Disk Pack Memory," *IEEE Transactions on Magnetics*, September, 1972 pp. 574-576.
10. Huffman, W. K. and H. L. Kalter, "An 8-K Bit Random Access Memory Chip Using a One Device FET Cell," *1973 IEEE International Solid State Circuits Conf. Proceedings*, pp. 64-65.

The economic implications of microprocessors on future computer technology and systems

by JAMES C. NELSON
Sperry Univac
St. Paul, Minnesota

INTRODUCTION (HOW DID WE GET HERE?)

At the turn of the last decade, 1970, the computer market was well structured. Almost 70 percent of the market was controlled by a single company, IBM, while (25 percent) of the remaining market was primarily controlled by four other companies. These companies produced the systems, software and hardware required for automation systems using second order suppliers to furnish a large proportion of the component parts. The largest second order suppliers were the semiconductor manufacturers who did an annual business of ($348 million/1970) annually producing the logic components for the required computer and peripheral system elements.[7]

As the semiconductor technology developed to permit integration of larger logical functions on a single semiconductor chip, a new low-cost computer industry evolved. This industry, the minicomputer industry, specialized in the production of dedicated computer systems for control applications in price ranges starting under $10,000.

Further development in the semiconductor industry centered around the memory requirements for computer manufacturers. It had been recognized much earliër that MOS technology had the potential of being more cost effective than the (at the time) low-cost core memory. However, the large investment in core memory tended to retard the development in semiconductor memory until such time as significant cost advantages were possible. The processes which eventually permitted the semiconductor memories to overtake the core memory were embodied in the 1103 semiconductor memory chip which eventually broke the core cost barrier in 1973-74.

Along with the development of the MOS process for memory came the development of MOS process to produce adding machines. The capability of placing all of the functions of an adding machine on one chip permitted an extreme cost reduction in this product, opening a new market to millions of potential users. Certain of the semiconductor manufacturers were quick to determine that the structure required to build a sophisticated adding machine was similar to that required for a simple computer, and development of such computers was begun. Initially this development was restricted severely in complexity by the semiconductor yields (a function of chip size, and thus complexity), as well as the limited number of pins on semiconductor packages then available. Initially, it is likely that semiconductor manufacturers saw the development of microprocessors as a means to increase sales of already existing semiconductor memories. However, it rapidly became apparent that these devices were capable of use in many new and old applications, such as intelligent terminals and numerical controllers for processes, which previously required assembly from discrete logic devices.

Further developments in this field involved the production of a microcomputer which was partitioned into bit elements of a minicomputer combined with a micromemory capable of sequencing simple microinstructions in order to affect, complex minicomputer instruction performance.

Another development expanded the four bit microcomputer capability to eight bits permitting these devices to efficiently operate for the first time with alphanumeric type codes, such as ASTIA. Thus, application of microcomputers to intelligent terminals and low speed data concentrators began to occur.

WHERE ARE WE NOW?

At the point of this writing, three major semiconductor manufacturers control over ninety percent of the microprocessing market. However, over twenty other manufacturers have announced competitive products in an attempt to penetrate this new market. The next year promises to be one of extreme competition between semiconductor-microprocessor vendors while they attempt to obtain market shares large enough to permit their continuation in this business area. Certainly many competitors will not survive in the keen competition which is about to occur.

In contrast with the initial microprocessor devices which were available, all of which began utilizing the more readily processed PMOS technology, microprocessors are now becoming available utilizing more exotic technologies which permit them to realistically compete with minicomputers in speed, as well as use in rugged environments. Among the technologies recently announced are; NMOS—which yields a factor of three in increased device speed of operation, CMOS—which permits wide temperature range for rugged applications, along with minimum power requirements, and single supply voltage, BIPOLAR TTL—which permits high speed and single power supply requirement, and INTEGRATED INJECTION LOGIC, which permits reasonably high speed, wide temperature range, and minimum power consumption.

Along with the development of these new technologies in microprocessors, better semiconductor process controls are beginning to permit larger quantities of logic to be placed upon a chip with reasonable yield. This improved process technique is resulting in the design of microprocessors with larger word lengths for applications where more powerful processing functions are required. At this time processors of 12 and 16 bit lengths have been announced by semiconductor manufacturers.

Although many manufacturers are attempting to build truly general purpose microprocessors for use in all applications, certain of the manufacturers have singled out specific applications areas, either based upon their processors design, or by design of chip sets for specific input/output applications. The function of peripheral control has been targeted by Rockwell while RCA is attempting to penetrate the entertainment market with a television add-on processor. Both Motorola and Rockwell are attempting to penetrate the communications market with specially designed microprocessor-modem parts, and Toshiba is attempting to win the automobile microprocessor control market with their wide-word, wide temperature range, 12 bit microprocessor.

In order to penetrate the microprocessor market, semiconductor firms have found that it is necessary to provide potential users with software capabilities almost as sophisticated as that provided by minicomputer firms. Most microprocessor manufacturers now have software assemblers and simulators available from timesharing services, or will provide Fortran decks for users who have their own computer systems.

Most of the microprocessor systems maintain their procedures (program) in read only memories, either fixed or programmable. In order to fix instructions within the ROMs, the user must either prepare data tapes for the semiconductor manufacturer or have available a means of programming their own semiconductor memories. In order to provide this function, and to permit checkout of microprocessor systems several manufacturers have made available low-cost ($2-3,000) hardware simulators which permit test of the system before the program is frozen in ROM or PROM.

At present, except in the case of National Semiconductor, few of the semiconductor microprocessor manufacturers provide assistance in application of their devices to actual function implementation. National is the only manufacturer that has indicated that it is in direct competition with the minicomputer manufacturer for application type business. All other microprocessor manufacturers indicate that they intend to operate as original equipment manufacturers of parts only.

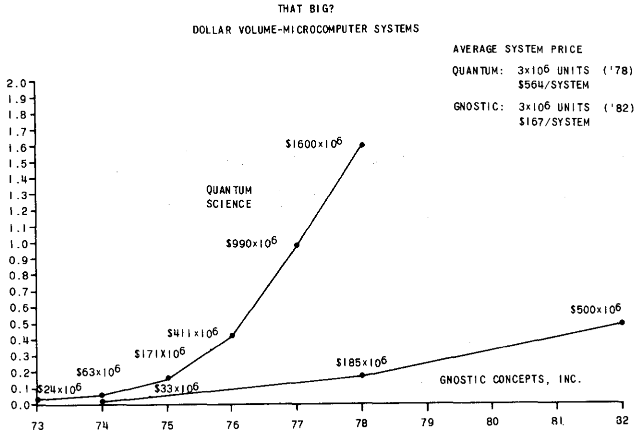


Figure 1—

TABLE I—Potential Microprocessor Control Market

Washing machine control	9 million per year
Dryer control	5 million per year
Electric range control	2.6 million per year
Television game interface	up to 12 million per year
Automobiles	8 million per year
Telephones	over 109 million

WHERE ARE WE HEADING?

Nonetheless, it is evident at this time that the microprocessor is headed for competition in two basic areas. First, it will penetrate the low end and eventually take over the minicomputer market,[2] secondly, it will generate many new high volume applications in areas previously unreachable by the computer industry due to the previously high cost of processor hardware.

There is no question that we shall see a marked change in the semiconductor business as a result of the evolutionary development of the microprocessor. For the first time the semiconductor manufacturer will have volume (end product) markets in automation application areas. Areas of business which could provide new markets for microprocessors directly to the semiconductor houses without intervention of computer manufacturers are shown in Table I below.[3]

Several market projections have been made during recent months. A summary of these market projections in dollar volume is given in Figures 1 and 2.[4-6] As indicated in Figure 1, market projections vary between $180 million in 1978 to $107 million in 1982. It is interesting to note that given an estimated demand of 3 million chip sets in the end projected year, the cost of chip sets by semiconductor manufacturers shows a cost varying between $36 and $90 per chip set.

Actual microprocessor system costs, projected in Figure 2 include memory and input/output units, in addition to the basic microprocessors. Average costs of these units vary from $167 to $564 in 1982 and 1978 respectively. One can only regard the large variation in these figures as an indication that the emerging microprocessor market is not yet well defined. No projection yet indicates the effect of the automobile market, estimated to begin using microprocessor product in 1980, and only one projection[4] indicates the in-house capability that will be developed by the present computer industry leaders in order to maintain value added in their peripheral and processor products.

One thing is patently apparent, and this is that there will be a fall-out of vendors of microprocessor products. No one winner will be evident, but rather several manufacturers concentrating on specific functional products will survive the shakeout, along with, perhaps, several general purpose device producers.

The effect on microprocessor users, such as numerical control manufacturers, and peripheral manufacturers will be the development of de facto interface standards that are enforced by the availability of only a few dominant types of microprocessor types to implement their device control. It is appealing to project that the survivor, and thus the interface definition will be determined by the best, most economical design available from the several semiconductor microprocessor producers. However, as already apparent in the industry, the surviving de facto standard is not always the best possible design, but rather an indication of the economic, marketing, and political elements existing at the time of the technology shakeout.

Projections of eventual costs of devices by semiconductor manufacturers indicate that piece parts of $10 per unit are possible for product volumes of 10,000 units or greater. Simple availability of such devices will force computer design engineers of larger systems to use these devices whenever possible because of economics. Thus the availability of microprocessors will affect even the architectures of larger machines in areas where speed is not of extreme importance.

Undoubtedly, because of cost advantage there will be a trend toward partitioning of those problems which will admit such decomposition into a series of serial processes which can be implemented by means of microprocessors with little sharing of the problem data base. (An example of such a partition for a radar processing function is given in Figure 3.) Some modification to current microprocessor architecture will undoubtedly occur to more readily permit sharing of certain memory banks by multiple microprocessors.

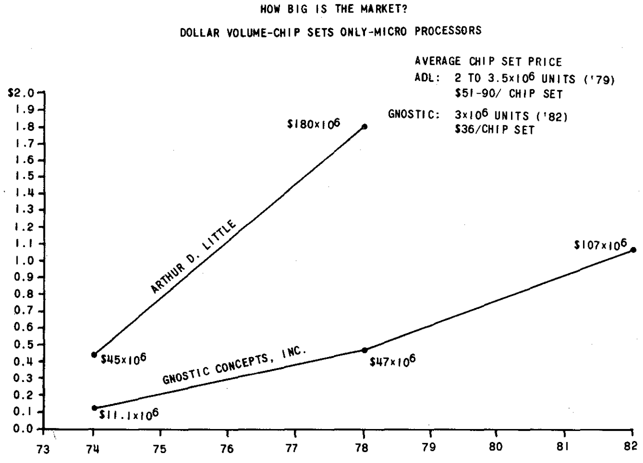


Figure 2—

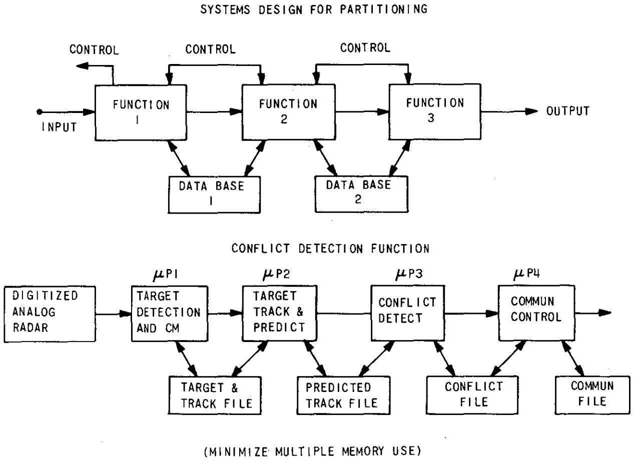


Figure 3—

Certainly in the near future there will cease to be any non-intelligent peripherals. The economics of the microprocessor dictates that all symbiont routines, previously held in the central processor for peripheral unit control, will gravitate to the microprocessor within each peripheral which controls its function and permits modification to affect various control disciplines.

Nevertheless, there will continue the need for the high performance general purpose processors of high complexity manufactured by the leaders of the processing industry. These processors will continue to be used for network control and as data base handlers, because the economics of the central data base will force this form of network organization.

SUMMARY

In summary, the microprocessor evolution will work great changes on the computer-automation field. Anticipated events within the next five years will include:

1. Growth of intelligent peripherals with microprocessing control, including even the simplest data entry device. Implementation of present software I/O symbionts into the fixed programs contained within the processors/controlling peripherals.
2. A renewed attempt to partition complex problems with little data base overlap so that a pipeline of microprocessors may be used to effect a more economic implementation of such problems.
3. Minimization of the actual control of processes by large scale processors, and a specialization of central

processor facility to provide network control and data base handling.

4. A fallout of microprocessor manufacturers with several specialized architectures and suppliers remaining to service different functions. (i.e., communications, entertainment, machine (auto) control, general purpose.)
5. As a result of the fallout, a de facto standard of devices and device interface which will force the large scale computer manufacturer to design to the needs of the microprocessor interface.

REFERENCES

1. "U.S. Markets 1971 Forecast," *Electronics*, pp. 63, January 4, 1971.
2. *Microprocessors and Microcomputers*, Auerbach Report S175.0000.100, September, 1974.
3. *Statistical Abstract of the United States*, U.S. Bureau of the Census, Washington, D.C., G.P.O., 1973.
4. *The Potential of Microprocessor Technology*, Arthur D. Little Impact Services, Norman S. Zimbel, September 1974.
5. *"Special Report, Microprocessors," Electronic News*, pp. 1ff, John Day and Martin Gold, May 27, 1974.
6. "Shipments of Microprocessors," *Electronic News Special Supplement*, pp. 4, August 12, 1974.

Innovations in the operation of future computers

by FREDERIC G. WITHINGTON
Arthur D. Little, Inc.
Cambridge, Massachusetts

EVOLUTIONARY TRENDS

Not many years ago the standard way of operating computers was in a single stream batch mode. The operator completely controlled the computer resource, setting up each job before its initiation, mounting the appropriate tapes and I/O media, initializing the machine, putting the appropriate program deck in the cardreader and so forth. The user interacted with the operator most often in a "closed shop" mode, giving the operator a run request from a card deck and tapes as needed, and leaving it to him to take care of applying the resources to the job.

Because of the waste of computer time involved in single stream batch operation, multi-programming became general. In the multi-program environment the roles of user and operator did not change in any relative way, but the role of the operator became impossibly difficult. He was still required to specifically allocate system resources to each job in the mix, but now he had to handle as many as five to ten jobs simultaneously in the system. He was also expected to see to it that regardless of the pattern of operation of each job the system resources devoted to each partition (particularly computer time and memory space) were efficiently used. This proved humanly impossible in most cases, and the computer operator's job became a very difficult and unhappy one.

Fortunately for the operator, demand was slowly growing for systems which would project the resources of the computer directly to the user. Beginning with computational time-sharing and evolving into production areas, modern systems provided the user with a computer that would respond when needed, both to compile new programs and to execute runs of programs in the library. These interactive concepts have by now spread over the entire spectrum of data processing. Remote data entry, remote batch processing, conversational program development, transaction processing and (in general) the association of integrated data bases with the organization's employees at every level have achieved general acceptance. A diversity of developments have been needed to make this possible:

- Virtual system management techniques have proved to be desirable. First, virtual memory management evolved to allocate the storage resource in a dynamic environment. Second, to a still controversial degree multiple operating environments will be provided through the use of virtual monitor techniques. Most recently came the concept of the virtual file, first in MULTICS and now in the IBM 3850, with more to come. The virtual file may prove to be at least as important as any of the other concepts of virtual operation, because it enables the system to provide data sets, program libraries and files dynamically in response to user demand without operator intervention.
- Data base management software is becoming mature in the sense of being reliable, versatile and easy to use in an on-line environment of multiple users. Particularly important is the development of user query languages, which not only permit users to inquire about the contents of files but increasingly permit them to develop and execute new procedures involving manipulation of file data.
- Communication processing subsystems including network definition languages, versatile access methods, and economical controllers have evolved to a degree that permits economical use of communications networks by users without specialized knowledge.
- A special case of interactive processing with particular implications for local operation is the self-diagnostic and remote maintenance capability being evolved by some of the manufacturers. Designed to reduce maintenance costs while providing increased systems availability to the user, remote maintenance will further reduce the need for people at the computer's location.

All of these developments reduce—sometimes nearly eliminate—the need for operators in the computer room.

Comparable developments are occurring in small systems, used both in association with large machines in networks and independently by small users. The dominant trend is toward the development of small, easy to use interactive systems typified by the new Honeywell Series 60/61 and the NCR SPIRIT. In these systems all processing takes place interactively. (Few small users were ever very interested in batch processing computers, because their small volumes of transactions had never made batch processing necessary. Using file cabinets full of ledger cards, accounting machines and telephones, the small users already had the full advantages of interactive processing—even though semimanual—and saw no need to put up with the time delays of the batch approach.) These new small systems, taking advantage of the low transaction volumes which enable the dedication of the whole system to one transaction at the time, are also causing the user to become the operator. The order entry clerk is automatically causing the machine to execute procedures to process each order as it is entered. Outputs are delivered to the receiving dock, the storeroom, or the production line. There are few changes of programs or disk packs, since the system is dedicated most of the time. Just as with the large systems, the need for an individual in the machine room to set up and execute each application program disappears.

The intelligent terminal represents a parallel development. An intelligent terminal is usually used in a hierarchical system where a central data base exists, performing some processing of the individual transaction, sometimes using local files applying only to the location where the terminal exists. Processing performed by intelligent terminals has also obviously left the control of the central system operator. He has no idea how much processing is taking place in the intelligent terminals of the system, nor any control over who does it, when, or in what manner.

IMPLICATIONS

In some respects these trends should be a cause of considerable concern. No single individual or group will have knowledge at the time of what the system is doing or for whom. If any failure, fraud, confusion or loss of data occurs, it will be far more difficult than in the past to identify the cause of the problem and to rectify it. The manufacturers face a formidable responsibility of providing fool-proof software which cannot be caused to fail by any combination of permissible actions on the part of the users, and which also provides all of the necessary audit trails, journals and control reports needed for the user to detect misuse and be able to recover from failure. The software must also provide responsive service to the user and run the system reasonably efficiently; experience shows that these objectives are not easily met either.

The user also has formidable challenges facing him. He must be able to identify which users consumed what resources, where bottle-necks in the system are occurring, and where any bugs in either the applications or the system itself are cropping up. In the old batch environment these monitoring functions were not too difficult (though in multi-programmed systems considerable complexities arose). In the new world of direct interaction with the user the system must monitor itself, which is easier said than done.

Perhaps more critical than the monitoring problems are those of data quality control, protection against system misuse and provision of adequate security and privacy of data. Even though the contents of the data base are being projected to many users in many locations, the user must somehow find a way to be sure that restrictions on access to the data are enforced and that accounting controls and protections against fraud are effective in all circumstances. In the past, with an orderly flow of batched transactions through manual check-points and with all files contained in the computer room, it was not difficult to provide these satisfactory controls. No comparably effective general methods have been developed for interactive systems; the user is challenged to develop specific methods which will be adequate in his individual case.

Presuming that the manufacturer and the user can deal adequately with these problems (and most manufacturers and users alike think that they can), the roles of the people as they interact with the systems will be changed in interesting ways.

The user becomes the operator of the system, in the old sense that he causes a run to be performed at a particular time requiring a particular set of resources. Except for the remaining periodic batch runs (such as the production of payrolls and month-end accounting reports) the remainder of the system's operations will be initiated by the user without the assistance or intervention of any central operating personnel. To a considerable degree he also becomes his own programmer; such tools as query languages merge the programming and command languages indistinguishably. The user finds it easy to prepare most reports and local, small programming runs; the application programmer loses his role as a provider of fast response special reports and small runs. However, the large batch jobs of the organization continue to be implemented by specialist application programmers; a large number of them are still needed.

The computer operator is affected more than the application programmer. His primary responsibility, used to be the application of system resources to every individual job; this role has disappeared altogether. To the degree that local batch runs remain the system still needs help in the mounting of forms on printers (until such time as all output is delivered remotely), in mounting tapes (until automated tape library machines come into wider use) and in performing similar supporting functions. Also, the operator will probably have to assist the machine in instances of conflict between jobs of equal priority, in resource allocation or schedule conflicts that are beyond the machine's ability to resolve for itself. The operator becomes, in a sense, the foreman who augments the intelligence of a group of cooperating robots. The operator also remains responsible for the master schedule: informing the machine of the necessity for performing regularly scheduled batch runs, making sure that the primary operating environments are available to users during the scheduled hours, and the like.

Finally, and perhaps most important, the operator becomes the custodian of the information resource. He must see to it that appropriate analytic studies are performed regularly, for purposes of accounting for usage and of tracking developing maintenance and bottle-neck problems. He must also (above all) see to it that the control and back-up procedures are adequately followed, that no illegitimate access to data is occurring via routes of access which he can monitor, that the appropriate journals of transactions and file images are maintained and safeguarded, and that in general all control procedures which require regular execution are followed. In this sense the operator has a more important job than he has ever had before, and the job is considerably upgraded from the applications-oriented one of previous generations. The operator becomes the overseer of the performance of the information systems resource; he becomes the conductor of the orchestra in which each user plays his preferred instrument.

User orientation in networking*

by ORRIN E. TAULBEE, SIEGFRIED TREU and JIRI NEHNEVAJSA

University of Pittsburgh
Pittsburgh, Pennsylvania

INTRODUCTION

Motivation

The virtues and drawbacks of computer networks are well known. A number of significant networks are already in existence,[1] and their effects and potential in the area of resource sharing are unquestionably important. Confirmation of this fact is indicated not only by the relevant literature on the whole,[2] but also by various NSF-sponsored research efforts[3,4] and special NSF initiatives.[5,6]

But real success in computer networking, totally aside from the question of economic justification, has been quite difficult to attain. This is not to say that important partial successes have not been scored. The ARPA Network, for example, represents a significant achievement in application of communication technology to networking.[7] But the communications subnet was not the only goal. The network was to be used—resources were to be shared.[8] Although some actual "users" have by now gotten involved, and the network implications previously described by Roberts promise considerable growth in usage,[9] indications are that the user was inadequately taken into account during initial network design stages. Only after the sophisticated communications network was established were detailed questions about network usability seriously raised and confronted. The latter were partly initiated by users themselves[10] and have also resulted in a network users group which meets periodically to discuss user problems and suggestions.

In stark contrast to the ARPA network, an Educational Information Network (EIN) was also started several years ago. It has left some lingering lessons behind.[11] EIN was also to enable sharing of computer resources. Much related discussion and planning resulted in an organizational framework within which a user could utilize the catalogued resources of member institutions. But the means of communication provided for EIN was not a telecommunications net. Users were to transmit their data by mail or courier! For this and other reasons, the users were obviously not attracted to EIN. Hence, in spite of the interest in sharing resources displayed by EIN member institutions, very little use was made of the network.[12]

In a sense, we have had a peculiar dichotomy between widespread interest in sharing of computer resources and adequately advanced technology to supply the essential telecommunication means and methods. We must endeavor to bridge the gap. To do so, we do not require substantial additional technological breakthroughs. Instead we need major advances in tailoring the available computer/communications technology to become more amenable to and suitable for human use. Regardless of how efficient a computer network might be, it must be approachable and usable in order not only to appeal to prospective users, but also to retain the current usership.

Therefore, user-orientation in computer network design and implementation is crucial. But how can it be achieved? First, it is necessary to recognize that the objective requires interdisciplinary attention. If the user is of genuine concern, psychological and sociological considerations must immediately enter the picture. Furthermore, network management must be involved. It must bring its directives and policies to bear. In past years, local computer center management frequently left design decisions affecting the user-computer interface to the whims and wishes of local computer programmers. That was serious enough in a local environment. But it must be viewed as intolerable in the context of a nationwide computer network.

Study context

This paper is one result of a recent NSF-supported study.[13] That study was purposely restricted to the major user-oriented considerations which should concern management. The broader roles in network management,[14] though recognized, were not addressed. Just how management itself is organized and distributed geographically was only questioned on the basis of effects (if any) on user services. Logically, the user sees (or should be able to see) management as centralized, regardless of actual organizational structure and location. Consistent with this, the currently popular attempt at stratifying networks into wholesale/retail functions,[15] is considered to be another manifestation of a kind of management which, on the one hand, might show a potential for facilitating services but which, on the other hand, the user "could care less about."

* Supported in part by National Science Foundation Grant GJ39989.

STUDY FRAMEWORK

Global view

There is a time for specialized or stratified approaches to studying a major problem area. This is especially true when a highly complex system is involved, for which some kind of total view may be overly ambitious and in fact counterproductive. In view of the significant progress that has been made, particularly with respect to network-related computer and communications technology, the time is now ripe for putting it all together on behalf of the consumer of network-based services.

We have therefore tried to confront the complexity of user-oriented network considerations at a more global view and higher level of abstraction. The intention was to

1. observe, conceptualize and organize the characteristics and problems of the computer network and its environment which are significant to the user, and
2. systematically utilize the results and other pertinent knowledge for the specification of those network features which should not only eliminate or at least alleviate user difficulties but actually provide a favorable network posture to the user.

All of this has been considered with the view that network management must ultimately be responsible. Much can be said about what is idealistically desirable. But, unless effective means and methods are made available for incorporating desired user-oriented features in network design and operation, and unless network management is properly motivated and has appropriate policies and procedures at its disposal, the desirable network characteristics will remain just that: desirable and seemingly unattainable.

Interdisciplinary approach

As long as a computer network is intended to attract and benefit the human consumers of its services, it must not only be designed and managed for technologically efficient operation; it must also capitalize on the expertise importable from other disciplines, particularly those which can facilitate human interactions or dealings with the network. The increasing sophistication of computer science and technology, as it is exhibited through the presently available as well as planned computer networks, justifiably enthralls computing professionals. However, we must not ignore the (probably) detrimental consequences for the present and prospective network user.

"Non-computer specialist" computer users have had and still do have a multiplicity of problems in trying to cope with past and present computer service modes. The very nature of a geographically distributed set of interconnected service centers presents the real danger of leaving the normal kind of user even further "removed" from what is going on and where to get help. Unless we are willing to suggest or concede that the consumer should learn to live at the mercy of hardware and software and be forced to adapt accordingly, we must deliberately start to take the requirements and preferences of users, whenever reasonable and feasible, into account in computer network design and subsequent management. To determine and understand what the user needs or wants, it is desirable to take advantage of the people-oriented disciplines of psychology and sociology.

Accordingly, our study has been carried out in a distinctly interdisciplinary manner. This should be apparent from the following description of the study procedure and various resulting behavioral considerations.

Three-phase procedure

Toward achieving the objective of developing a framework of consumer-oriented considerations in network design and management, a procedure was followed which consisted essentially of three phases:

1. Descriptive modeling
2. Structured reasoning, and
3. Policy mapping

These are successively characterized below.

Descriptive model

Without any externally imposed network design constraints, a suitable computer network configuration had to be specified as an exemplary model. A realistic and useful network configuration, based both on precedent-setting examples among existing networks as well as on conceptions of what a nation-wide network for science might be like, had to be hypothesized. The study results should then have particular reference to the modeled network.

To hypothesize a network of national scope, without the benefit of any surveys or estimates on who might participate, from where and what for, obviously requires a number of assumptions. These are characterized in conjunction with Figure 1. As that figure shows, the conceptualized model configuration is decentralized. Its nodes are considered to be located at preselected, geographically distributed sites and they collectively involve some set of heterogeneous, medium- to large-scale computer systems. The internode communications facilities are assumed to be adequate for accommodating the reasonable requirements of the network users.

This leads to the next level of our descriptive model: the user population which is necessary to render a network viable. It is depicted by the arbitrary pattern of *U* symbols in Figure 1. With our primary objective of consumer-oriented network design and management, this level is of course at the heart of concern. What are the possible problems confronting the average user who is trying to access and utilize network facilities from one of the indicated locations? They are not insignificant; they may be very frustrating if not altogether intolerable. The user is faced with a broad spectrum of questions or potential problems, ranging from whether and where the resource of interest is available, to how to get access to it and utilize it, and how to take care of various administrative matters such as authorization of system use. Throughout this range of user problems, a variety of psychological considerations enter the picture. These are dealt with more closely in a later section.

Having settled on a particular technological configuration for a model network, and having superimposed a required user population for whom, after all, the network services should be intended and designed accordingly, we are ready for the third and final level of our model. It is precipitated by asking whether the network capabilities in terms of hardware and software are viewed to be in themselves sufficient for serving the user successfully. In spite of the aforementioned laudits for technology, the answer is a definite no. Whether we like it or not, people are still required—even in the context of a sophisticated computer network. Hence, as indicated in Figure 1, the user may need to acquire assistance or advice from geographically distributed network staff members or perhaps even other users of the network. Furthermore, the users as well as staff members may have to directly or indirectly relate to or communicate with network management, whatever form the latter might take.

Consequently, we actually find that all of

1. a user population,
2. a supporting staff, and
3. the management personnel

are involved in what can be viewed as a "social system" superimposed on the computer network. The sociological considerations relevant to this level of our descriptive model will be highlighted later.

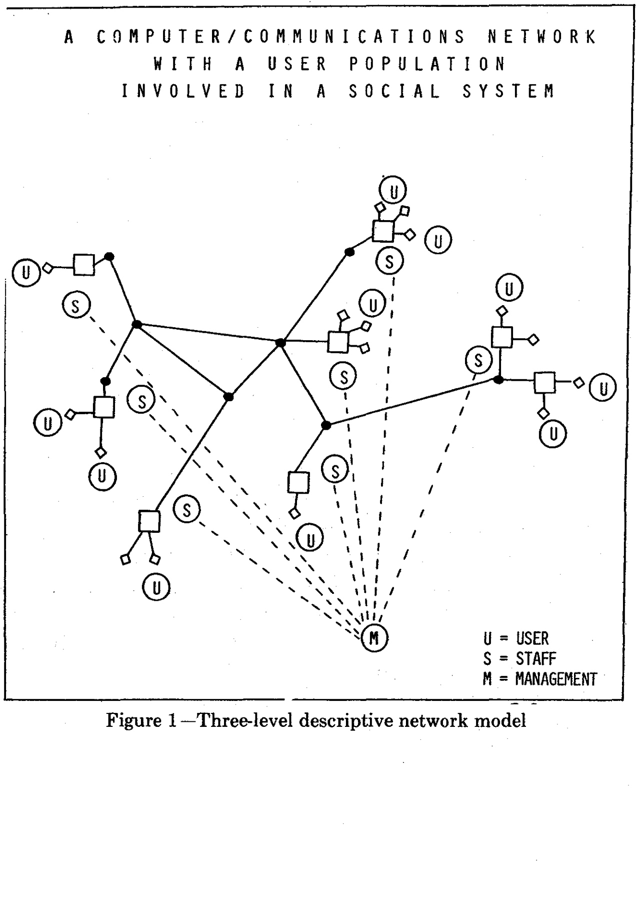

A COMPUTER/COMMUNICATIONS NETWORK
WITH A USER POPULATION
INVOLVED IN A SOCIAL SYSTEM
U
S
M
U = USER
S = STAFF
M = MANAGEMENT

Figure 1—Three-level descriptive network model

It should be apparent from the description thus far that the nature of this study is indeed interdisciplinary. In addition, since all of the above should be understood, directed and facilitated by network management, appropriate policies and procedures must be established.

Structured reasoning

With reference to the descriptive network model, specific network features now had to be identified along with any corresponding recommendations for management. To accomplish this in a deliberate and thorough manner, an organized structure was imposed on the search for or reasoning out of those network characteristics which are deemed to be consumer-oriented.

From the outset, this structured reasoning phase stratified the network considerations[16] into those pertaining to

1. Usability,
2. Sociability, and
3. Accountability

The term "usability" was employed to encompass those network features and capabilities which have a significant bearing on the user's fundamental inclination and ability to interact with the network to take advantage of its available resources. "Sociability" was selected to include those network features and capabilities which facilitate or enable the establishment of various cooperative links or colleagueships, in the interest of better utilizing available resources. Thirdly, we used "accountability" to refer to those network features and capabilities which permit the user's direct or indirect interaction with management in accounting for network access, use, billing and other related problems.

The three categories of consideration are clearly interrelated in a number of ways. Nevertheless, each area is significant in itself and warrants some separate attention. In this paper, however, emphasis can only be given to the first two because of space limitations. The third category, accountability, and implications of all three categories for network management, will be treated in a separate paper.

Our "structured reasoning" entailed one additional step: the establishment of a framework according to which the desired network features could be determined more systematically. This framework is an array. For each of the usability and sociability (as well as accountability) strata, the framework was employed, as will be seen from Tables I and II, toward finding a desired set of factors in response to each of the following three questions which are indicated by row labels of the array:

1. What is the availability of appropriate resources?
2. Is the user-network interface suitable?

Table I—Usability Factors

NETWORK CHARACTERISTICS AND FEATURES	USER KNOWLEDGE AND PREPAREDNESS	DIAGNOSTIC OR FACILITATING FACTORS
1. Availability of Appropriate Resources:	1. On Resources:	1. With Regard to Resources:
(a) Hardware Facilities	(a) Awareness of Available Resources	(a) Market Survey
(b) Software Capabilities	(b) Willingness to Find Out about Them	(b) Publicity
(c) Data Bases	(c) Need to Use Them	(c) Regular and Prompt Updating of Information about Resources Available
(d) Directory to Resources	(d) Other	
(e) Other		(d) Other
2. Suitability of User-Network Interface:	2. On the Interface:	2. With Regard to Interface:
(a) Terminals	(a) Experience with Different Interface(s)	(a) Training both On-line and Off-line
(b) Interactive Languages	(b) Willingness to Learn How to Use It	(b) Preparation and Distribution of Appropriate Documentation
(c) Procedures and Rules	(c) Willingness to Study Documentation	
(d) Documentation	(d) Other	(c) User Feedback Leading to Potential Redesign
(e) Other		
		(d) Other
3. Level of Network/Interface Performance:	3. On Performance:	3. With Regard to Performance:
(a) Accessibility	(a) Realistic Expectations	(a) Continuous Evaluation
(b) Reliability	(b) Reasonable Tolerance	(b) Security Checking
(c) Responsiveness	(c) Willingness to Accept Blame when Appropriate	(c) Quality Control
(d) Data Security		(d) Other
(e) Other	(d) Other	

3. Are the network resources as well as interface performing properly?

Further structure is imposed via the dichotomy portrayed by the first two columns of the array. In spite of the frequently mentioned advocacy of user-orientation in this paper, that is not to imply a totally one-sided relationship between user and network. The user does also have certain, minimal obligations. While the network must supply the necessary resources, the user must at least show enough interest to be minimally informed and prepared. Thus, for each network characteristic to be listed in the first column of the array, one or more corresponding obligatory factors in behalf of the user are implicitly sought in the second column.

Policy mapping

If we can successfully identify the significant consumer-oriented network features, then the final question to be addressed is: what should network management do about them, or what policies and procedures should they adopt and carry out?

This third phase of our study procedure can be viewed as a kind of policy mapping. The third column of Figure 2 can be used to define this concept. If the first two columns portray the features to which the network and the user should respectively contribute, what are the factors, or means and methods, which are available toward diagnosis/reinforcement/facilitation of the desired features on the part of management? Furthermore, management must be cognizant of and apply suitable policies and procedures to assure success. Thus, while the factors to be listed in the third column are responsive to "what" is necessary, certain policies and procedures are required to say "how" it is to be done.

PSYCHOLOGICAL CONSIDERATIONS

Usability array

Network usability factors pertain to the user-network interaction. Collectively they represent a kind of idealized profile of man-machine partnership or rapport. The network is expected to provide all those features which carry psychological implications for user attraction to and satisfaction by the network. The user, on the other hand, should be obligated to do comparatively little: gain minimal knowledge and preparedness and display a reasonable attitude toward the network.

These considerations are brought out in more detail through the structured reasoning phase, resulting in the usability array displayed in Table I. The latter is not intended to be exhaustive and entries are only indicated in capsule form.

From the standpoint of interest and welfare of the user, the network is unlikely to attract attention unless it has some appropriate resources to offer. These might include special hardware facilities or software packages which are not locally available, and perhaps one-of-a-kind data bases to which access is sought. And to assure awareness by the user of the available resources, a very important resource in itself is a (preferably on-line) directory to the repertoire of network facilities. But all of that is to no avail if the prospective user does not have a need (or at least a natural curiosity) to become aware of what the network can offer, and to expend the effort toward finding out.

Utilization is promoted if the user or potential user is aware of the resources available and how to use them. This means that adequate steps must be taken to publicize and announce what resources are available, when they are available, where they are available, and how the user can take advantage of this opportunity. Such publicity must be complete and up-to-date in that it includes the latest changes about each of the resources.

In order to keep the resources of the network attuned to the needs of the user a mechanism for surveying user wants and experiences in utilization must be established. This must be accomplished on a regular basis so that network performance quality can be maintained through improvement of existing resources or by re-design.

Assuming that those first hurdles are overcome and a user-network match-up is potentially in the making, the details of actual interaction with the network gain prominence. Does the network support the kind of hardware terminals, e.g., graphic devices, which the user wants or needs? Is the interactive language, or set thereof, too heterogeneous and confusing? Are the required interaction procedures too cumbersome, and is the related documentation out-of-date or lacking in clarity? Answers which are unfavorable to the user will surely tend to lessen if not eliminate enthusiasm for the network. But even if interface features are favorable, the user is again obliged to play his part. Experience with other interfaces will normally help, although prior conditioning to certain characteristics and expectations can also negatively affect a user's view of a new facility. In any case, the user must demonstrate a willingness to learn whatever is required in order to interact with the network in question. This task must not be too difficult; otherwise, the user will give up.

An individual desiring to use the network's resources should be able to take advantage of an orientation and training program to satisfy his need for basic information about the network. Such a training program may be conducted in both on-line and off-line modes. Distribution of documentation may take place at an off-line training session. However, adequate documentation should be readily available at other times as well.

Finally, with reference to Table I, the basic network resources and interface characteristics may exist and appear to be attractive, but their respective performances may be lacking. A much longer list of performance factors, viewed as desirable from the user's standpoint, could be listed. While some of them, such as reliability, may be deemed more important than others, the overall "usability" of the network is significantly affected by its performance profile. However, as before, the user must do his part. He must be realistic and reasonable in what he expects and what he is willing to tolerate. This is of course to some degree influenced by the user's past experience (and conditioning to other computer services) and resulting understanding of and appreciation for what is realistically possible.

Selected special concerns

Not all the important features are apparent from or even listable in Table I. Instead, some result from various collective treatments or abstractions on those features which are listed. A few of these are highlighted in Table I.

Attracting users

What is it about the psychology of a user which causes him/her to be attracted to a particular service, such as a computer network? Actually both psychology and sociology are involved in the process. To begin with, somebody has to be innovative enough to try it regardless of whether he knows of anyone else having done so. Then, in trying the network, he finds himself faced with an assortment of network features like those listed in Table I, and the questions relating thereto. After a trial period, some particular mixture of favorable characteristics may tip the user's "scale" in favor of network-related success, as op-

Table II—Sociability Factors

NETWORK CHARACTERISTICS AND FEATURES	USER KNOWLEDGE AND PREPAREDNESS	DIAGNOSTIC OR FACILITATING FACTORS
1. Availability of Appropriate Resources:	1. On Resources:	1. With Regard to Resources:
(a) Staff Specialists for Various Resources	(a) Awareness of Available Human Resources	(a) Identification of Specialists Among Other Users
(b) Specialists Among Other Users	(b) Willingness to Locate Them	(b) Selection of Staff Specialists
(c) Directory to Human Resources	(c) Need to Use Them	(c) Publicity on Availability of Specialists
(d) Other	(d) Other	(d) Other
2. Suitability of Interpersonal Interface:	2. On the Interface:	2. With Regard to Interface:
(a) Special Communication Software/Hardware	(a) Interpersonal Communications Experience	(a) Training and Scheduling of Specialists
(b) Interpersonal Communication Procedures	(b) Willingness to Communicate	(b) Opportunity for Regular Face-to-Face Meetings between User, Staff, and Management
(c) Documentation	(c) Willingness to Learn Special Procedures	(c) Insuring Sensitivity to User Concerns
(d) Other	(d) Other	(d) Other
3. Level of Human Network/Interface Performance:	3. On Performance:	3. With Regard to Performance:
(a) Access to Specialists	(a) Tactful Communication	(a) Continuous Evaluation
(b) Human Responsiveness	(b) Demonstrated Appreciation	(b) Incentives Program for Staff and User Specialists
(c) Cooperation	(c) Appropriate Credit	(c) Instill in Staff that Network is a Means to an End for the User rather than an End in Itself.
(d) Other	(d) Other	(d) Other

posed to failure. The mixture does of course vary from user to user, subject to personal priorities, preferences, and again, prior experience.

The important byproduct of successful user experience with the network is that a satisfied user is probably the best catalyst for attracting additional users. The attraction of users in this manner entails basically three steps:

1. Observing or being informed that another user (e.g. colleague) has been successful in using the network and likes it,
2. Actually bringing himself to try it,
3. Experiencing that an adequate combination of network features (in Table I) is favorable and that his required personal efforts (also Table I) are not excessive.

Thus the user has essentially been given positive reinforcement for expectations, personalized to himself, but originating from promises or projections based on someone else's experiences.

Informing users

Another concern, which cuts across the considerations in the second column of Table I, is how much information the user actually needs, or is forced to assimilate, in interacting (or learning how to interact) with the network. The user who really wishes to know all about available resources, access procedures, interactive languages etc. is probably the exception rather than the rule. Instead, at any particular time of being motivated to access the network, the user wants to know only what he needs for successful interaction. And he wants that information to be made available expeditiously and unambiguously.

Ideally, the network should enable the user to learn, on the one hand, as little as necessary and, on the other hand, as much as desirable. Over a period of time, this will cause the user to develop more insight than if initial experiences with the network required the user to learn a multitude of details about the total network. Opportunities for learning as much as desired, when desired, should however be available. The extreme of inundating the users must not be replaced by the extreme of withholding pertinent information or perhaps cloaking it in technical jargon which only specialists understand.

Maintaining users

After a user has been attracted to the network and, then, has been adequately but not excessively informed about the network, what is to make sure that he does not turn into a non-user again? Aside from any economic considerations, the user who needs the particular resources available can be expected to remain a customer either until the network services and performance deteriorate below a certain, personal threshold of tolerance, or until a competitive computer service with better performance becomes available.

So, the key to success in maintaining users lies not only in assuring that those characteristics which originally attracted the user are continued at the same or higher levels of performance; it also requires a certain amount of innovativeness toward improving services, without generating serious disruptive effects. As the user gets more experience and gains a better understanding of what the network can and could do, he probably does not want to stay in a networking environment which is comparatively static. Thus, aside from pure economic and administrative concerns, the user is likely to remain satisfied if the network achieves a reasonable balance between service quality and innovative hardware/software development toward modifying/improving network capabilities.

SOCIOLOGICAL CONSIDERATIONS

Sociability array

The social system of users, staff and management, when superimposed on the technological network, plays a significant role in actually enabling or facilitating network use. The "sociability" concept is therefore not addressed for its own sake, but because of its bearing on "usability", described earlier. The primary concern here is with overcoming the depersonalization which naturally arises in networking where, by design, a network as a "system" is automated to the maximum extent possible consistent with economic and reliability considerations. This means that communication between users, staff and management must be natural and free flowing. Thus, ability to communicate should be an important factor in selecting staff and management personnel. Beyond this, mechanisms to promote communication should be established.

Analogous to what was done with usability in Table I, sociability factors are portrayed in capsule form in Table II. It can be seen that the main emphasis is on users "socializing" with the network staff or other users. Network management is assumed as implicit in or behind the staff representatives of the network.

It is interesting to observe that many of the factors listed in Table II are analogous to those in Table I, except that now a human network of resources is that object of attention of the user. This means that on the network side appropriate staff specialists must be made available and identifiable by means of a suitable personnel directory. In addition, for certain highly specialized projects and personal resources accessible via the network, other users might be the best (if not the only) available sources of information. Given their willingness to participate, they can significantly supplement the network's repertoire of human resources. As was true for the hardware and software resources, the user must now play his/her part by gaining awareness of the human resources and then, when necessary, calling on them.

But that is more easily said than done. Given an up-to-date directory of network specialists, the regular telephone call can be employed for a limited amount of interpersonal communication. However, in a reasonably sophisticated network, various (computer) network-supported modes can be expected and are preferred for certain tasks. Voice conversations can of course be maintained in addition to on-line, computer-based communication.

As was the case for the user-network hardware/software interface, the hardware/software-supported interface for interpersonal interaction must be suitable for the user. Otherwise, if it presents too much of an obstacle, he will become frustrated and refrain from any further attempts to get (remotely located) human assistance in the network.

Thirdly, with reference to Table II, the human resources aspect in the network must perform satisfactorily. If specialists are announced to be available at scheduled times, the user should be able to rely on that. Furthermore, once the user-specialists communication link is established, it is not too much to ask that the specialist be as patient, responsive, and cooperative as is reasonably possible. Managers of local computer centers know only too well what problems arise from lack of adequate and sincere user-orientation on the part of staff members. This possibility is most likely to be compounded in attempted communication between a user and a remotely located resource person.

Selected emphasis areas

Within the social system consisting of users, staff and management, it is desirable to give special attention to each of the following pairwise combinations in the network environment: user-user, user-staff, staff-staff, management-management (perhaps multi-level), user-management, and staff-management. The following two sections briefly address only the first two of these; the others will be considered in a separate paper.

Users and users

One important reason for one user's wish to gain the assistance of another was already indicated: the case in which the latter is perhaps the best or only authority on the use of a particular piece of hardware or software or data base. But other reasons may exist.

This leads to a potentially significant sociability feature in a computer network. If the network can maintain some kind of accessible record of usage patterns and related user problems, as exhibited or experienced by individual users, then the capability to search such records could lead to user-user match-ups and subsequent interactions.

Of course the above would not be carried out without expressed user permission. User privacy and the proprietary nature of his software or data must clearly be protected. However, if and when a user determines that the time is ripe for him to share information about either his newly developed data bases and debugged software packages or his other substantive network-related activities and experiences, he should be able to so indicate, with the assurance that he will be given appropriate credit and can turn this accessibility off when he desires.

Users and staff

One of the serious problems in user-staff interaction is overcoming the certain "in-group barrier" which tends to develop and surround network personnel. The latter are very likely to evolve an esprit de corps, based on the significance and attractiveness of belonging to the network organization, such that they become rather self-centered and regard the struggling users out there as almost alien.

This is, in a sense, not too surprising in view of the normal constituency of a network staff: computer programmers, information retrieval specialists, computer systems analysts and others with similar occupational titles. These persons are naturally inclined to view their common objective, namely the computer/communications network, as the only thing of importance.[17] But regrettably this is done strictly from their occupational vantage points, without attempt to reach out and learn to appreciate what the service consumer at the other end of the network is really trying to do. These users do not normally regard the network as all-absorbing. They tend to be persons from industry, business, government or academia who only wish to use the network as a means to an end, not an end-in-itself.[18]

That gap between staff and users must be bridged. And once the staff members can be encouraged, enticed or ordered to develop greater interest in and understanding of user problems, then they can become more receptive and responsive to user inquiries for assistance. Needless to say, as was indicated previously, the user must in turn demonstrate not only respect, but also the tact and appreciativeness which makes for much better two-way communication among people.

CONCLUSION

A review of the literature on computing indicates a plethora of material on networking. But in this literature there is a paucity of publications dealing with users in relation to a network. The current paper, and the study from which it is derived, contributes to the literature on networking because:

1. It emphasizes user concerns and focuses on certain features which make the network consumer- and service-oriented.
2. It highlights the importance of an interdisciplinary approach in studying one aspect of networking.
3. It displays one structured method of attacking such a complex system.

REFERENCES

1. Farber, D. G., "Networks: An Introduction," *Datamation,* April 1972, pp. 36-39.
2. Blanc, R. P., I. W. Cotton, T. N. Pyke, Jr. and S. W. Watkins, *Annotated Bibliography of the Literature on Resource Sharing Computer Networks,* NBS Special Publication 384, September 1973.
3. Dorn, W. and M. Robbins, *Alternative Approaches to the Management and Financing of University Computing Centers,* University of Colorado, NSF Grant No. GJ-32368.
4. Weingarten, F. W., N. R. Nielson, J. R. Whiteley, and G. P. Weeg, *A Study of Regional Computer Networks,* University of Iowa, Partially supported by NSF Grants GJ-27723, GJ-27724. February 1973.
5. Aufenkamp, D. D., "National Science (Computer) Network," *Networks for Higher Education,* Proc. of EDUCOM Spring Conference, April, 1972, pp. 29-35.
6. Greenberger, M., J. Aranovsky, J. L. McKenney, and W. Massey, *Networks for Research and Education: Sharing Computer and Information Services Nationwide,* MIT Press, Cambridge, 1973.
7. Frank, H., R. E. Kahn and L. Kleinrock, "Computer Communication Network Design—Experience with Theory and Practice," *SJCC,* Vol. 40, 1972, pp. 255-270.
8. Roberts, L. G. and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *SJCC,* Vol. 36, 1970, pp. 543-549.
9. Roberts, L. G., "ARPA Network Implications," *EDUCOM Bulletin,* Vol. 6, No. 3, Fall 1971, pp. 4-8.
10. Pickens, J. R., "Evaluation of ARPANET Services, January-March 1972," UCSB Computer Systems Lab, RFC #369 July 25, 1972.
11. Le Gates, J., "The Lessons of EIN," *EDUCOM Bulletin,* Vol. 7, No. 2, Summer 1972, pp. 18-20.
12. Denk, J. R., "Exchange of Applications Programs for Education—A National Stalemate," *Interface* Vol. 5, No. 1, February 1971, pp. 11-21.
13. Taulbee, O. E. and S. Treu, *User-Oriented Computer Network Management,* Dept. of Computer Science, University of Pittsburgh, Report Under NSF Grant GJ-39989, 1975.
14. Stefferud, E., "Management's Role in Networking," *Datamation,* April 1972, pp. 40-42.
15. Stefferud, E., D. L. Grobstein and R. P. Uhlig, "Wholesale/Retain Specification in Resource Sharing Networks," *Computer,* August 1973, pp. 31-37.
16. Taulbee, O. E. and S. Treu, "Usability, Sociability and Accountability in Computer Networks," Bulletin of 1974 Computer Science Conference, February, 1974, Detroit, Michigan (Abstract Only).
17. Thompson, Victor A., *Modern Organization,* Alfred Knopf, 1961, pp. 16 ff.
18. Lefton, Mark, "Client Characteristics and Structural Outcomes," in Rosengren, William and Mark Lefton, *Organizations and Clients,* Charles E. Merrill, 1970.

A deterministic analytic model of a multiprogrammed interactive system

by SAMUEL T. CHANSON
Purdue University
W. Lafayette, Indiana

and

DOMENICO FERRARI
University of California
Berkeley, California

INTRODUCTION

The analytic models used to evaluate the performance of multi-access computer systems are generally queueing models. Their ability to represent, often adequately, these systems is due both to the congestion situation created by the processes competing for a system's resources and to the variability of the workload, whose parameters can be much more concisely characterized as stochastic variables than by some deterministic representation. Thus, deterministic models have been only applied in extremely simple, first-approximation studies (see, for example, Hellerman and Smith[1] and Fenichel and Grossman[2]), or in the analysis of some resource management algorithms (e.g., the problem of optimal schedules discussed in Chapter 3 of Coffman and Denning[3]). This paper presents a non-queueing model of a multiprogrammed interactive system. The model is substantially more sophisticated and more accurate than the deterministic models previously proposed, but retains the characteristics, common to all deterministic models, of greater mathematical simplicity and better understandability with respect to queueing models of comparable accuracy. Even though the equations of the model derived in the paper for an XDS 940 installation cannot probably be applied to other systems, the authors feel that the approach can be used for most systems to obtain quick and reasonably accurate estimates of some performance indices.

THE APPROACH

The system to be modeled in the study reported here was an XDS 940 time-sharing computer with a primary memory of 64K words, a drum, a movable-arm disk and a number of interactive terminals. The system is multiprogrammed but its memory is not automatically managed; since a process must completely reside in memory before its execution is started, its information is swapped in at the beginning of an interaction (unless the process is already in memory) and is swapped out, if room for another process is needed, at the end of the interaction. Each process is given control of the CPU for a maximum amount of time, called a *quantum*; when the quantum expires, the CPU is switched to another process (if there is at least one being ready to run) and the interrupted process joins a queue of ready processes. The 940 has two quantum lengths; a process is allocated a short quantum first, and, if this is exceeded, then the process receives a longer quantum.

In the model, a process (and hence the workload, defined as the ensemble of processes using system resources during a given interval of time) is characterized by the durations of its three possible states (see Kimbleton and Moore[4]); *active* (running), *ready* (waiting for the CPU), and *blocked* (waiting for, or using, a secondary memory or I/O device). Since this *process-state model* is not load-independent, in that the duration of the three types of periods for a given process generally depends on the behaviors of the processes competing with it, a simpler process-state model will be introduced. This is the one characterizing the given process in the so-called *homologous* uniprogramming environment, i.e., when it runs on the same system and the same configuration in uniprogramming mode; evidently, in this case a process can only be in one of two states, namely, active or blocked.

The performance indices selected for the study were the CPU utilization ρ and the mean response time $\overline{\text{RES}}$. These indices will be expressed as functions of the parameters which characterize the workload, first in the multiprogramming environment and then in the homologous uniprogramming environment.

The derivation of the model is based on several assumptions, the most important of which are summarized here. In deriving most of the equations, the workload is assumed to be *uniform*, i.e., to consist of identical processes. (When the identical processes in a uniform workload have very narrowly distributed state durations, the workload is said to be *homogeneous*.) The assumption of uniform workload will be discussed in a later section. Another assumption which seems to be, and in fact is, very unrealistic is the *lockstep mode* (i.e., fixed order) in which the processes are supposed to be executed (see also Kimbleton and Moore[4]). This assumption, however, has been found to produce tolerable errors in many

TABLE I—Symbols Used in the Text

N	Number of non-dormant processes in the system ($N=1, 2, \ldots$)
N_a, N_r, N_{blk}	Total numbers of active, ready and blocked periods
N_{tty}, N_{drm}, N_{dsk}	Total numbers of teletype, drum and disk I/O operations
A, R, B	Durations of active, ready and blocked periods (in *ms*) (subscripts indicate position in temporal sequence or process they refer to; when the number of processes is important, they appear as $A(N)$, $R(N)$, $B(N)$)
B_{tty}, B_{drm}, B_{dsk}	Durations of blocked periods due to teletype, drum, and disk I/O
L	Quantum length (in *ms*)
s	Probability of having to swap in a process's pages
Tc	Swap-in time for C pages (in *ms*)
C	Mean number of pages needed by a process to run
M	Primary memory capacity available to user processes (in page frames)
ρ	CPU utilization
RES	Response time (in seconds)
$\bar{X}$	Mean of random variable X
$\lceil X \rceil$	Smallest integer not lower than X
$p(X=x)$	Probability that random variable X takes the value x

cases; some experimental data confirming this conclusion will be presented in Table VI. Other simplifying assumptions made consist of neglecting the overhead and I/O activity generated by the operating system; swapping-in and swapping-out times were incorporated into the active periods of the process involved, and all I/O operations were considered as generated by user processes. Finally, only one quantum length was assumed, for simplicity.

The simulator used to validate the model is described in detail in Reference 5. Its design, features and degree of detail are similar to those of Nielsen's time-sharing system simulator.[6] The validation of the simulator was based on measurement data collected on the 940 system during normal operation.

THE BASIC EQUATIONS

In this section, the two performance indices we have selected, namely, CPU utilization ρ and mean response time $\overline{\text{RES}}$, are expressed in terms of the parameters of the system's workload. The symbols that will be used in this and in the subsequent sections are listed, together with their meanings, in Table I. It should be noted that the workload parameters in Table I characterize the behavior of the set of processes which constitute the workload, or of each single process in it, over a time interval called *total elapsed time*, which is measured from the instant the N processes in the workload enter the system (their arrivals are assumed to be simultaneous) to the instant the execution of the last process terminates. Also the two performance indices will be computed over that interval.

CPU utilization is defined as the fraction of the total elapsed time during which the CPU is active. In a uniprogramming environment, we have

$$\rho = \frac{\bar{A}(1)}{\bar{A}(1)+\bar{B}(1)} . \tag{1}$$

In a multiprogramming environment, the CPU is active for a total time period equal to $N_a\bar{A}(N)$. The total elapsed time would be equal to the duration of the mean interval between the start and the termination of one process, which is

$$\bar{D} = \frac{1}{N}\,(N_a\bar{A}(N)+N_r\bar{R}(N)+N_{blk}\bar{B}(N)), \tag{2}$$

if the processes all ended at the same time. Since this is never the case, due to the fact that most of the resources in a computer system are non-shareable, the actual elapsed time is always longer than $\bar{D}$. However, if the workload is relatively homogeneous (in particular if the first and last few active and blocked periods are not disproportionately long), then the error incurred by taking $\bar{D}$ as the total elapsed time is small, provided that N_a is large enough. Hence, ρ can be approximated by

$$\rho \cong \frac{N_a\bar{A}(N)}{\bar{D}} = \frac{NN_a\bar{A}(N)}{N_a\bar{A}(N)+N_r\bar{R}(N)+N_{blk}\bar{B}(N)} , \tag{3}$$

which, when $N=1$, reduces to (1).

The values of ρ given by (3) were compared with simulation results for a number of realistic workloads and in all cases it was seen that the error made by using (3) never exceeded a few percent, and that the accuracy of (3) increases as N (hence also N_a) becomes larger.

The response time for an interactive user can be defined as the interval between two consecutive blocked periods due to teletype I/O (ignoring echoes). Note that the usual assumption is made here that computing is finished before the response starts being output. For $N_{tty} \gg 1$, the total number of interactions is practically N_{tty}. Hence, taking the average over the total elapsed time, we obtain

$$\begin{aligned}\overline{\text{RES}}(N) = \frac{1}{N_{tty}}\,(N_a\bar{A}(N)+N_r\bar{R}(N)+N_{drm}\bar{B}_{drm}(N)\\ +N_{dsk}\bar{B}_{dsk}(N)).\end{aligned} \tag{4}$$

Note that the lockstep assumption has not been invoked in deriving (3) and (4), and that (4) is fully valid even for non-uniform, non-homogeneous workloads.

EXPANDING THE BASIC EQUATIONS

Among some of the parameters in Table I, there exist the following two relationships, which can be derived from their definitions:

$$N_{blk} = N_{tty}+N_{drm}+N_{dsk}, \tag{5}$$

$$\bar{B}(N) = \frac{1}{N_{blk}}\,(N_{tty}\bar{B}_{tty}(N)+N_{drm}\bar{B}_{drm}(N)+N_{dsk}\bar{B}_{dsk}(N)). \tag{6}$$

Since all four parameters in (5) appear in the basic Equations (3) and (4) (whereas this is not the case for (6)), one of the twelve parameters in equations (3) and (4) is dependent on some of the others and eleven are the independent parameters. These parameters can be thought of as characterizing the workload being multiprogrammed and are therefore called the *multiprogramming parameters.* We shall now try to express each one of them, with the only exception N, which we assume to be known, as a function of the process-state parameters which characterize each process in the homologous uniprogramming environment and are therefore called the *uniprogramming parameters.* Both types of workload parameters are functions of system parameters. However, most of the system parameters influencing the uniprogramming parameters (e.g., CPU speed, primary and secondary memory speeds) are likely to remain unchanged, or to change very few times, during the lifetime of the system. For straight-line programs (in which all loops are executed a number of times specified within the program), the values of the uniprogramming parameters can be obtained directly from the listings of the programs if the speeds of the hardware units of the system are known. Otherwise, some of the parameters must be measured in the homologous uniprogramming environment, provided that the system being modeled does exist. It should be observed here that these measurements are to be performed only once and that, unless the system has been designed with measurement in mind, it is easier to measure these workload parameters than the performance indices themselves (whose measurement cannot be used to predict the effects of system or workload modifications). We shall now see how each one of the ten multiprogramming parameters we are interested in can be computed.

N_{tty}, N_{drm}, N_{dsk}

Assuming that all I/O requests are initiated by the users, N_{tty}, N_{drm}, and N_{dsk} are workload parameters not affected by the system. They can either be measured or, in the case of straight-line programs, be obtained by counting the number of teletype, drum and disk I/O operations respectively in the listings of the N programs. The identical-process assumption is not necessary to obtain these values.

TABLE II—Validation of Equation (7)

Workload Type	N	ρ	N_a/N_{blk} Equation (7)	N_a/N_{blk} Simulation	Percent Difference
QED	5	0.1	1.0	1.0	—
QED	9	0.6	1.03	1.03	—
M2	5	0.6	1.24	1.20	+3.5%
M2	9	0.8	1.27	1.23	+3.2%
M1	5	0.92	1.81	1.73	+4.6%
M1	9	0.97	1.90	1.84	+3.2%
QSPL	5	0.99	4.07	3.77	+7.5%
QSPL	9	1.00	4.2	3.95	+6.5%

N_a

If N is so large that CPU utilization is very near to 1, it is highly probable that any CPU time request greater than one quantum will result in more than one active period. In fact, if A_i is the CPU time request of a process at a certain instant, then, with a probability approaching unity as N increases, the number of active periods resulting from this request is $\lceil A_i/L \rceil$.

Since for most systems the normal workload tends to keep high CPU utilization, we can approximate N_a by

$$N_a = N_{blk} \sum_i p(A = a_i) \lceil a_i/L \rceil, \tag{7}$$

where i runs over the entire histogram of A.

Table II compares the values of N_a/N_{blk} given by (7) to those obtained by simulation under various workloads, which are described in the Appendix. The accuracy of Equation (7) depends on the type of workload as well as on the value of ρ. It is better for those workloads where most or all of the CPU demands can be satisfied in one time quantum (e.g., the QED workloads). For the same type of workload, the higher ρ (i.e., the larger N) is, the higher the accuracy. However, those workloads for which ρ is low usually have relatively small CPU demands and hence Equation (7) may still give a good approximation (see the first rows of Table II).

N_r

For identical processes running in lockstep mode, a ready period is present whenever one or more of the following conditions are satisfied:

1. a CPU time request is greater than L and there is at least one other process waiting to use the CPU;
2. the duration of a blocked interval is less than the sum of the active durations of the rest of the N processes in the same period (see the timing diagram in Figure 1 (a)):

$$B_i < (N-1)A_i; \tag{8}$$

3. it is both (see Figure 1(b))

$$B_i > (N-1)A_i \quad \text{and} \quad A_{i+1} > A_i. \tag{9}$$

Given the histograms of A(1) and B(1), N_r can be calculated from the three conditions above as

$$N_r = N_{blk} \sum_j p(A(1) = a_j) \{ (\lceil a_j/L \rceil - 1) + p(B(1) < (N-1)a_j) + p(B(1) > (N-1)a_j) p(A(1) > a_j) \}, \tag{10}$$

where j runs over the entire histogram of $A(1)$.

Even though the lockstep assumption is generally not satisfied in a real-life situation, simulation has shown that, for a relatively homogeneous environment, the estimates of N_r given by (10), which are based on this assumption, are reasonably accurate (see Table 111).

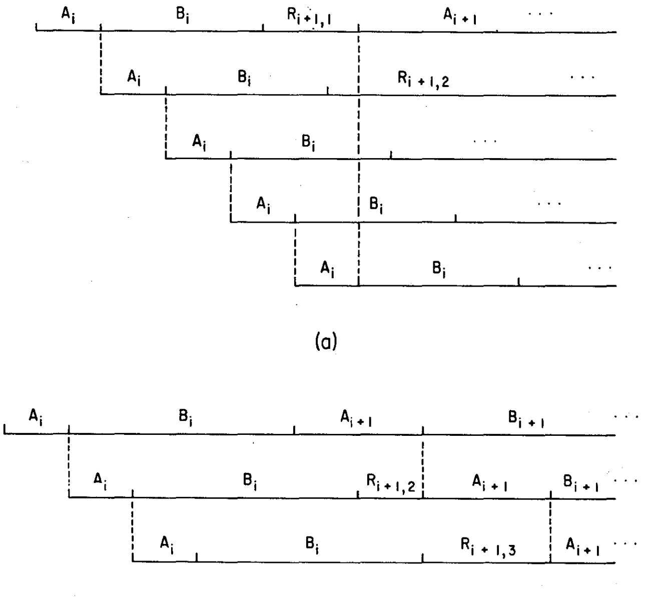


Figure 1—Timing diagrams illustrating the creation of ready periods when (a) condition (8) is satisfied ($N=5$) and (b) condition (9) is satisfied ($N=3$).

$\bar{A}(N)$

The mean active duration of the N processes in the system is affected by variations of N in two conflicting ways. First, as N increases, $\bar{A}(N)$ tends to decrease since it is more likely that there is a ready process waiting for the CPU at the end of a quantum. At the same time, however, since the time needed to swap in a process's pages at the beginning of an active period is considered part of $A(N)$, and since the probability of swapping increases as N increases, $\bar{A}(N)$ also tends to increase with N. Thus, we can write (without making use of the lockstep assumption)

$$\bar{A}(N)=\bar{A}(1)\frac{N_{blk}}{N_a}+sT_C. \tag{11}$$

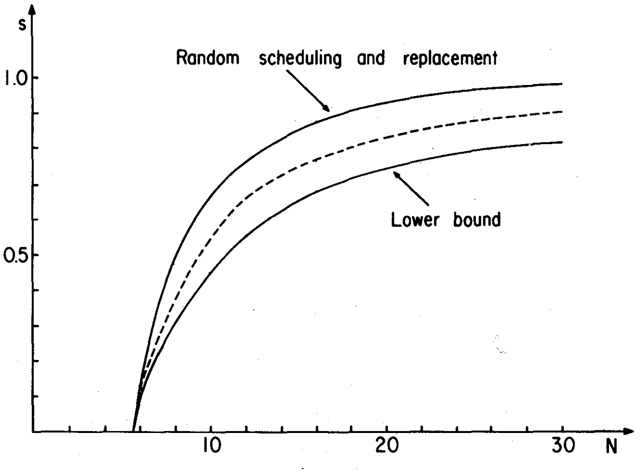


Figure 2—The curve of the probability of swapping ($M=32$, $C=6$).

The probability of swapping, s, depends on N, M and on the page requirement of each process. In the long run, and for N identical processes in the system, we have

$$s\begin{cases}\geq\dfrac{CN-M}{CN} & \text{for } N\geq\left\lceil\dfrac{M}{C}\right\rceil,\\[2ex] =0 & \text{for } N<\left\lceil\dfrac{M}{C}\right\rceil.\end{cases} \tag{12}$$

The value of s is likely to be near its upper bound when random scheduling and random page replacement policies are used. The curve of s as a function of N, obtained by simulation using these policies, is plotted, together with the lower bound given by (12), in Figure 2 for $M=32$ and $C=6$. The actual curve lies somewhere between these two limits, and may be approximated by the middle curve (dotted in Figure 2). The average time T_C to bring C pages into primary memory is the product of C by the average time required by the swapping drum to transfer a page (neglecting the latency time in both the one-page and the C-page cases). The two errors introduced in (11) by estimating s and T_C as described above have caused the results obtained from (11) to differ from those of our simulation experiments by less than 10 percent.

$\bar{R}(N)$

To estimate $\bar{R}(N)$ accurately, it is necessary to know for each process the uniprogramming durations of active and blocked periods in their chronological order, i.e., the sequence A_1, B_1, A_2, B_2, . . . , rather than the two distributions of $A(1)$ and $B(1)$. Each A_i is incremented by sT_C (see Equation (11)) and, for $A_i>L$, it is replaced by $(A_i/L-1)$ L and $(A_i-L(A_i/L-1))$ (i.e., it is assumed that every CPU request exceeding L will be interrupted at the end of the quantum). Let R_{ij} be the duration of the i-th ready period of the j-th process. Even though processes are assumed to be

TABLE III—Validation of Equation (10)

Workload Type	N	N_r/N_{blk} Equation (10)	Simulation	Percent Difference
M1	7	1.715	1.75	−2.0%
M1	15	1.844	1.88	−1.9%
M2	5	0.71	0.66	−7.6%
M2	11	1.0	1.015	−1.5%
QED	8	0.35	0.41	−14%
QED	15	0.87	0.985	−12%
QSPL	7	4.03	4.1	−1.7%
QSPL	15	4.15	4.22	−1.6%

identical, each one of them will experience different ready periods.

Assuming lockstep execution, we have for $j=1, 2, \ldots N$:

$$R_{i+1,j}=\begin{cases}\max\{(j-1)(A_{i+1}-A_i), 0\} & \text{for } B_i \geq (N-1)A_i \\ \max\{(j-1)A_{i+1}+(N-j)A_i-B_i, 0\} & \text{for } B_i < (N-1)A_i.\end{cases} \tag{13}$$

Equations (13) have been obtained by using timing diagrams such as those in Figures 1(a) and 1(b). As shown in Table IV, the lockstep assumption makes Equations (13) much less accurate than the others presented so far. However, they always overestimate $\bar{R}(N)$, whereas Equation (10) always underestimates N_r, as seen in Table III. Thus, since N_r and $\bar{R}(N)$ always appear as a product in the basic equations, the two errors partially compensate each other.

$R(N)$ may also be derived by the same approach used to obtain Equation (10). This allows $R(N)$ to be computed with the knowledge of only the two distributions of $A(1)$ and $B(1)$ at the expense of accuracy. $A(1)$ is to be modified by adding sT_C to each a_i to form a_i'. The a_i's greater than the quantum length L are dropped with the sum of their probabilities given to $a_i'=L$. Then

$$\begin{aligned}\bar{R}(N) \approx \sum_j p(A'(1)=a_j')\Big\{ & p(B(1)\geq (N-1)a_j') \\ & \cdot p(A'(1)>a_j')\left(\frac{N-1}{2}\right) \\ & \times \Big(\sum_{a_k'>a_j'} (a_k' p(A'(1)=a_k'))-a_j'\Big) \\ & + p(B(1)<(N-1)a_j')\left[\frac{(N-1)}{2}\,(a_j'+\bar{A}'(1)) \right. \\ & \left. - \sum_{B_l<(N-1)a_j'} B_l p(B(1)=B_l)\right]\Big\},\end{aligned} \tag{13'}$$

where j runs over the entire (modified) histogram of $A'(1)$. The error was found to be about 20 percent.

Note that, if $N_a \gg N_{blk}$ (i.e., in the case of a heavily computebound workload, with $\bar{A}(1) \gg L$), a process has to wait almost at every interaction for the other $N-1$ processes.

TABLE IV—Validation of Equation (13)

Workload		$\bar{R}(N)[ms]$		Percent
Type	N	Equation (13)	Simulation	Difference
QED	5	3.62	3.44	+5.1%
QED	13	377	349.68	+8.0%
QSPL	5	2450	2213	+10.2%
QSPL	11	6440	5634	+14.2%
M1	5	1100	969.5	+13.5%
M1	13	3540	3092	+14.2%
M2	5	600	541	+10.5%
M2	13	1352	1176	+15%

TABLE V—Validation of Equation (16)

	$\bar{B}_{drm}(N)$ $[ms]$		
Workload		Simulation	
N	Equation (16)	$K=5$	$K=500$
3	31.4	32.39	30.76
5	39.0	38.39	37.59
9	55.2	57.85	53.39
13	69.5	71.82	71.21

Thus, the approximate equation

$$\bar{R}(N) \cong (N-1)\bar{A}(N) \tag{14}$$

can be used.

$\bar{B}(N)$, $\bar{B}_{drm}(N)$, $\bar{B}_{dsk}(N)$

For those interactive users who do not make excessive use of secondary memory devices, $\bar{B}_{tty}(1)$ is at least one order of magnitude greater than both $\bar{B}_{drm}(1)$ and $\bar{B}_{dsk}(1)$. Therefore, $\bar{B}(1)$ will reflect mostly $\bar{B}_{tty}(1)$, and since think time in front of a teletype is likely to vary very little with N, $\bar{B}_{tty}(N)$ will approximately equal $\bar{B}_{tty}(1)$. Thus, we can write

$$\bar{B}(N) \cong \bar{B}(1), \tag{15}$$

and the analogous approximate equalities for $\bar{B}_{drm}(N)$ and $\bar{B}_{dsk}(N)$, because of the assumption of very light secondary memory usage, which is equivalent to the one of empty disk and drum queues.

When this assumption is not satisfied, we have to resort to the ordering of the active and blocked periods again. Assuming identical processes, let $B_{ij}(N)$ be the duration of the i-th blocked period of the j-th process in the multiprogramming case $(j=1, 2, \ldots N)$, and let B_i be a short-hand notation for $B_{i1}(1)$. Then,

$$B_{ij}(N) = (N-j)B_{i-1}+jB_i-A_i, \tag{16}$$

with $j=1, \ldots N, i=2, \ldots K$ and $K=\min\{x \mid A_x \geq (N-1)B_x\}$.

Note that K is the number of I/O operations in a burst of I/O activity and that index i counts the operations in the burst from its beginning. Equation (16) is based on the lockstep assumption applied to a steady-state condition $(K \to \infty)$ and considers a burst of I/O operations to a single I/O device, which can be represented by a single-server queueing system with FCFS policy. Most secondary memory I/O devices employ a multi-queue scheme with FCFS or some other policy for each queue. Generalizations of (16) to cover these cases are presented in Reference 5.

It should be noted that Equation (16) (or its generalizations) are in fact used to calculate $\bar{B}_{drm}(N)$ and $\bar{B}_{dsk}(N)$; these values, together with that of $\bar{B}_{tty}(N)$ derived from the distribution of console times, which we assume to be known, and with those of N_{drm}, N_{dsk} and N_{tty}, can then be introduced in (6) to obtain $\bar{B}(N)$.

In spite of the crude assumptions made to derive (16), the results of its usage are quite satisfactory even for small values of K (see for example their comparison with simulation results for $\bar{B}_{drm}(N)$ in Table V).

TABLE VI—Comparison Between Real and Nearly-Equivalent Workloads

Workload		CPU Utilization			Mean Response Time [ms]		
			Nearly-Equivalent Uniform			Nearly-Equivalent Uniform	
Type	N	Real	Analytic	Simulation	Real	Analytic	Simulation
D1	7	0.854	0.834	0.862	758.5	714.1	719.0
D2	9	0.905	0.875	0.866	1011.7	1004.0	939.0

REAL WORKLOADS AND THE ASSUMPTION OF IDENTICAL PROCESSES

The equations derived in the previous sections are applicable only to workloads consisting of identical jobs. This section outlines the procedure for transforming a workload consisting of N non-identical jobs into a uniform workload. Two workloads are said to be *nth-order equivalent* with respect to a given system and a given set of performance indices if the first n moments of the distributions of the selected performance indices they produce are the same. For instance, if CPU utilization and response time are the selected indices, a first-order equivalent of a given workload is a workload which produces the same mean values of CPU utilization and response time. Thus, we are interested here in constructing a set of N identical processes which is first-order equivalent to the real workload, or, more precisely, in determining its characterization in terms of uniprogramming parameters.

By looking at the equations derived in the previous section, in particular at Equations (13) and (16), it is easy to realize that, for two workoads to be first-order equivalent, the condition that they have the same values of N, N_{drm}, N_{dsk}, N_{tty}, and the same distributions of $A(1)$, $B_{drm}(1)$, $B_{dsk}(1)$, $B_{tty}(1)$ is not generally sufficient; indeed, in the general case, this condition (to be called here the condition of *near equivalence*) must be supplemented with the one that the orderings of the active and blocked periods be such that they produce the same values of $\bar{R}(N)$, $\bar{B}(N)$, $\bar{B}_{drm}(N)$ and $\bar{B}_{dsk}(N)$ for both workloads.

We shall now see how a uniform workload which is first-order nearly equivalent to a given real workload can be obtained. Since deriving a nearly equivalent workload is much easier than determining an equivalent one, the question of whether and to what extent the former can be used in our model instead of the latter will then be examined.

If the real workload is composed of N straight-line programs, then it is not too difficult to determine the numbers of teletype, drum and disk operations. It is also a matter of simple calculation to find $\bar{A}(1)$, $\bar{B}_{drm}(1)$, and $\bar{B}_{dsk}(1)$, assuming that all the I/O requests are explicitly issued by the users. $\bar{B}_{tty}(1)$ depends on the users and may be obtained either by monitoring the system, if it exists and is measurable, or from published distributions of console times.

If the programs are not straight-line, then the values of the above parameters must be obtained by direct measurement. It is again emphasized here that the measurement needs to be performed only once and that it is usually easier to measure these workload parameters than the performance indices themselves.

With these data, N identical programs having the same proportions of I/O operations as the totality of the N processes in the real workload and the same values of $\bar{A}(1)$, $\bar{B}_{drm}(1)$, $\bar{B}_{dsk}(1)$ and $\bar{B}_{tty}(1)$ can be conceptually created. The problem of selecting the order of the various operations in this workload, which is nearly equivalent to the real one, presents itself naturally at this point. In our experiments, the values of the uniprogramming durations were either kept constant or varied only slightly above or below the mean so that the ordering became less important. In these cases, for non-I/O-bound workloads (i.e., when N_{blk} is small with respect to N_a), the order of blocked periods does not significantly affect performance estimations (see Equations (14) and (15)). Also, for I/O-bound workloads, where $N_{blk} \approx N_a$, the order is again unimportant since the programs tend to become homogeneous. It was found that by using arbitrary orderings in a relatively homogeneous, uniform, nearly-equivalent workload, the analytically computed results never differ from the corresponding simulation results by more than 10 percent (see some sample comparisons in Table VI).

In the next sections, some applications of the model to performance estimation, performance prediction and system tuning (in particular, the optimum choice of quantum length) will be illustrated.

PERFORMANCE ESTIMATION

Let us consider a uniform workload consisting of $N=5$ processes identical to the one schematically represented in Table VII. (Note that the short console times are due to the fact that the XDS 940 deals with teletype I/O on a break-character rather than on a line-by-line basis.) Let M be 32 pages, C be 3 pages and L be 250 ms. Using the hardware speeds of the XDS 940 system and the data in Table

TABLE VII—Description of a Process

i	$A_i(1)$ [*ms*]	$B_i(1)$
1	50	Read one page from disk
2	60	Console time (1.5 *s*)
3	80	Read two pages from drum
4	20	Console time (1.5 *s*)
5	100	Write three pages onto disk
6	30	Write one page onto drum
7	70	Console time (1.5 *s*)

VII, we can derive the distributions of $A(1)$, $B_{drm}(1)$, $B_{dsk}(1)$ and $B_{tty}(1)$, and obtain the following results.

$$\begin{array}{lll} N_{tty}=15 & \bar{B}_{tty}(1)=1500\ ms & \bar{A}(1)=58.7\ ms \\ N_{drm}=10 & \bar{B}_{drm}(1)=25\ ms & \bar{B}(1)=710\ ms. \\ N_{dsk}=10 & \bar{B}_{dsk}(1)=200\ ms & \end{array}$$

From Equation (15) we have $\bar{B}_{tty}(5)\cong\bar{B}_{tty}(1)=1500$ ms, and, since the value of K in Equation (16) is zero, there are no bursts of I/O activity. Thus, $\bar{B}_{drm}(5)\cong B_{drm}(1)=$ 25 ms, and $\bar{B}_{dsk}(5)\cong\bar{B}_{dsk}(1)=200$ ms. Since $\lceil M/C\rceil=11$ is greater than N, Equation (12) shows that the probability of swapping is zero. Also, since Equation (7) gives $N_a=35$, from Equation (11) we have $\bar{A}(5)=58.7$ ms.

To calculate $\bar{R}(5)$, note that, at the beginning, the four processes following the first have to wait 50, 100, 150 and 200 ms respectively. Since $A_1=50$ ms and $B_1=100$ ms, we have $(N-1)A_1>B_1$ and equation (13) allows us to compute the durations of the five ready periods which are produced: $R_{21}=100$ ms, $R_{22}=110$ ms, $R_{23}=120$ ms, $R_{24}=130$ ms, $R_{25}=140$ ms. Since $A_2=60$ ms and $B_2=1500$ ms, we have $(N-1)A_2\leq B_2$, and Equation (13) gives $R_{31}=0$, $R_{32}=20$ ms, $R_{33}=40$ ms, $R_{34}=60$ ms, $R_{35}=80$ ms. The durations of the subsequent ready periods can similarly be determined, their mean turns out to be $\bar{R}(5)=127$ ms and $N_a=29$.

We now use the basic Equations (3) and (4) to compute $\rho=0.338$ and $\overline{\text{RES}}(5)=533$ ms. A simulation experiment with the same workload produced the values $\rho=0.328$ and $\overline{\text{RES}}(5)=515$ ms. Thus, the model overestimated both indices by approximately 3 percent.

PERFORMANCE PREDICTION

The model can be very easily used to estimate the effects on performance of certain changes in the system. For instance, taking the same workload considered in the previous section (see Table VII) and assuming that the drum is replaced by one twice as fast, B_3 and B_6 will be 15 ms and 10 ms instead of 30 ms and 20 ms respectively. Console times are not likely to change appreciably. Similarly, a change of CPU speed would affect all the A_i's in Table VII by the same percentage.

As another example, if the capacity of main memory is reduced from 32 pages to 10 pages of the same size, $\lceil M/C\rceil$ becomes less than N, and, since primary memory cannot hold everything, swapping is likely to occur. The diagram in Figure 3 allows us to estimate the probability of swapping. As a result, $\bar{A}(N)$ and $\bar{R}(N)$ increase, and so does the mean response time.

QUANTUM LENGTH OPTIMIZATION

The quantum length may have a sizable influence on performance and can be changed rather easily in a well-written operating system. Thus, it is in principle feasible to vary the quantum length dynamically to optimize performance. While this topic still requires a considerable amount of research, the choice of the quantum length which optimizes system performance under the normal workload of an installation is an important aspect of system tuning. The model developed in this paper can offer some help in this direction. For a given system and with a given workload, assuming $N_a\gg N_{blk}$ (so that Equation (14) holds), and using Equation (11), we get from (4)

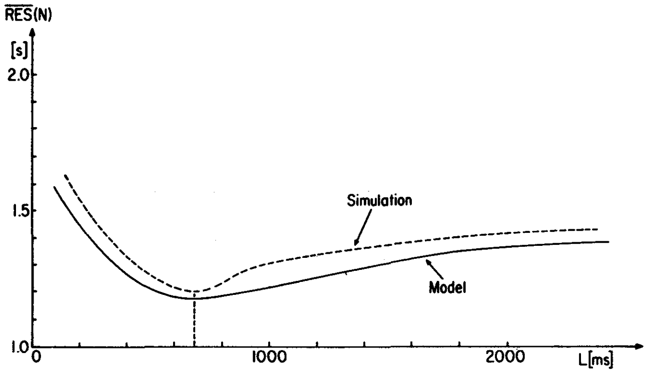


Figure 3—Mean response time vs. quantum length (workload M1, $N=7$).

$$\overline{\text{RES}}(\text{N})=C_1+C_2N_r\left(\frac{1}{N_a}+1\right), \qquad (17)$$

where C_1 and C_2 are constants.

To minimize mean response time, the value of $N_r(\frac{1}{N_a}+1)$ must be minimized. Since both N_a and N_r are non-negative, the mean response time will be at a minimum if $N_r=0$. This is what happens when every process which emerges from the blocked state always finds the CPU available, so that there are no ready periods. This situation can be achieved when ρ is very low. When both the system hardware and the workload are fixed, the term $N_r(\frac{1}{N_a}+1)$ can be changed by varying the quantum length L, since both N_a and N_r are affected by it. As L decreases, N_a will increase but so will N_r. Therefore, given a workload, there may be a value of L which minimizes the mean response time. Figure 3 shows the effect of changing the quantum length in a version of the XDS 940 having only one time quantum. The workload used is the one called $M1$ with $N=7$ (see the Appendix). The diagram shows that a quantum length around 700 ms will minimize $\overline{\text{RES}}(N)$ for this workload. It can be observed that, when L approaches zero, the curve goes up rather steeply. This is due to the sharp increase in swapping activity. We have found that, for most cases, $\overline{\text{RES}}(N)$ remains quite high if L is not greater than about 90 percent of all CPU time requests. As L is increased, $\overline{\text{RES}}(N)$ may rise slightly as some processes may have to wait for longer periods when the CPU is executing processes which have large CPU time requests. Beyond a certain value of L, when the quantum length is greater than any of the CPU requests, $\overline{\text{RES}}(N)$ will no longer be affected by variations of L and the curve will flatten out to a horizontal line.

CONCLUSION

An analytic non-queueing model of the XDS 940 multiprogrammed interactive system has been presented. In order to construct the model, a number of approximations had to be introduced, some of which are quite unrealistic. However, simulation results (obtained from a simulator validated by measurement) have shown the model to be reasonably accurate for the purposes of a number of performance evaluation studies. Some of the possible applications of the model have been briefly described.

The equations which constitute the model of the 940 system reflect the peculiarities of that system and cannot be used to describe any other system. However, it is felt that the approach is applicable to a large class of multiprogrammed systems, especially to those whose workloads are not too far from being uniform or, even better, homogeneous, since under these conditions the model is much easier to construct and much more accurate. An investigation of the applicability of the approach to the modeling of systems different from the XDS 940 is the most obvious next step of this research.

ACKNOWLEDGMENT

The research reported in this paper was supported in part by the Joint Services Electronics Program under Contract USAF-F44620-71-C-0087.

REFERENCES

1. Hellerman, H. and H. J. Smith, "Throughput Analysis of Some Idealized Input, Output and Computer Overlap Configurations," *Computing Surveys*, Vol. 2, No. 2, June 1970.
2. Fenichel, R. R. and A. J. Grossman, "An Analytic Model of Multiprogrammed Computing," *AFIPS Conference Proceedings*, Vol. 34, SJCC, 1969.
3. Coffman, E. G. Jr., and P. J. Denning, *Operating Systems Theory*, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.
4. Kimbleton, S. R. and C. G. Moore III, "A Probabilistic Framework for System Performance Evaluation," *Proceedings ACM-SIGOPS Workshop on System Performance Evaluation*, Harvard University, April 1971.
5. Chanson, S. T., *Workload Characterization of Multiprogrammed Computer Systems*, Ph.D. Thesis, University of California, Berkeley, 1974.
6. Nielsen, N. R., "The Simulation of Time-Sharing Systems," *Communications of the ACM*, Vol. 10, No. 7, July 1967.

APPENDIX

The types of workloads used to obtain the validation results presented in the tables of this paper are described here. They were constructed from experimental data measured in the Berkeley XDS 940 installation.

Uniform Workloads

Three types of uniform workloads were considered.

QED

These are predominantly interactive workloads (QED is the name of the 940's interactive test editor). Ninety percent of the durations of active periods are less than 10 ms, and the mean active time is about 6 ms. The mean blocked time is dominated by the terminal time and is about 1.5 s. Only 5 percent of all I/O requests are for the drum (with mean duration 28 ms) and 5 percent are for the disk (with mean duration 200 ms).

QSPL

These are mostly compute-bound workloads (QSPL is the name of the 940's major higher-level language). The mean active period is about 2.2 s and the mean blocked time is about 370 ms. Only 4 percent of all I/O requests are for the terminals (with a mean blocked time of 0.75 s).

Mixed

Several workloads having characteristics in between those of QED and those of QSPL workloads have been constructed. They are denoted by the symbol MX, where X is an integer. The greater the value of X, the closer the workload to the QED type. $M1$ and $M2$ have mean active times equal to 480 ms and 125 ms, respectively, and mean blocked periods of 400 ms and 500 ms, respectively.

Non-Uniform Workloads

These, denoted by the symbol DX, were mixed workloads composed of non-identical processes.

Experimental testing in programming languages, stylistic considerations and design techniques

by BEN SHNEIDERMAN
Indiana University
Bloomington, Indiana

BACKGROUND

In the early stages of the development of high-level languages, radically differing alternatives were often promulgated. Each language had a dedicated corps of adherents who advocated the primacy of their facility. Turbulent debates among the protagonists were a common affair at conferences and in the trade journals. Now as the field matures, the vehement discussions have subsided and there is a widespread recognition of the usefulness of a variety of languages. Even the proponents of a single universal language have softened their tone and have accepted the multiple language condition.

New proposals for algorithmic languages offer only slight variations, and much effort has been devoted to standardization. Simultaneously, there has been a proliferation of modest language extensions producing a conglomeration of dialects of the accepted standard. Widely varying languages are still developing, but primarily for specific problem domains, such as data description, data manipulation, or artificial intelligence research.

As the issues become more subtle, it is no longer acceptable for developers and implementors to make highly subjective and personalized statements concerning the worthiness of a particular language feature or stylistic technique. The rampant proliferation of new dialects or entirely new languages is counterproductive since it limits the sharing of programs. A concerted effort must be made to ensure that new features, dialects, languages, and techniques are truly improvements. Control cards for operating systems and utility programs could also be improved by proper experiments with users.

As computer utilization becomes more widespread, large numbers of programming amateurs in diverse disciplines will demand facilities which are simple to use. Professional parametric users, such as bank tellers and reservations clerks, with a minimum of training will also have to be accommodated. Since the background and orientation of these users is profoundly different from that of the programming language designer, experimental techniques must be devised to guide the designer to the optimum language specification.

Although Dijkstra explicitly stated that computer programming was primarily a *human* activity as early as 1965,[1] it was not until the publication, in 1971, of Gerald Weinberg's text *The Psychology of Computer Programming*[2] that this notion was widely recognized. This stimulating and insightful work set the stage for research into the human factors in programming. Weinberg's text concentrates on defining the programming task in the context of the professional environment and promotes the notion of "egoless programming teams." This team organization concept may be contrasted with the "chief programmer team" strategy advocated by IBM.[3] Experimental comparison of interactions in these personal organization strategies would be an intriguing task for social psychologists.

Other sections of Weinberg's book concentrate on individual personality factors, training, and motivational factors. Although the conference reports of the ACM Special Interest Group on Computer Personnel Research describe initial steps, much more research needs to be done on the psychological make-up of programmers. Fortunately, psychologists have begun to study programming behavior as an aspect of problem solving.[4]

Training and teaching of programming has long been of interest to academically oriented researchers, as witnessed by the papers presented at the annual conferences of the ACM Special Interest Group on Computer Science Education. Programming has only recently become a subject for related disciplines such as educational psychology.[5]

Although experimentation in all of the above mentioned areas would undoubtedly be welcome, the focus of this paper is on experiments in programming language features, stylistic considerations and design techniques.

Research in programming language design

The volume of written material on programming language design is enormous. Thorough comparative surveys can be found in the work of Sammet,[6] Higman[7] or Elson[8]. Detailed remarks by the designers occur in the classic reports on languages such as FORTRAN, COBOL, ALGOL 60, ALGOL 68, or PASCAL. Standardization reports on FORTRAN and COBOL also provide certain insights. Recently the overall topic of programming language design has been discussed by Hoare,[9] Wirth[10] and Carlson.[11] All of these works are based on the observations of individuals or small groups. The controversy over structured programming is closely related to program design issues. Although much has been written on this topic, the experimental evidence for eliminating the GOTO statement in favor of the three Bohm and Jacopini[12] structures has never been collected or reported. While individual experiences are useful,[13,14] they do not provide meaningful results to make judgments for the entire community of programmers.

Two groups of researchers have recognized the usefulness of studying non-programmers in the hopes of developing languages more closely conforming to "natural" thought processes. Sime, Green, and Guest[15] describe a fascinating experiment on non-programmers to determine which of two conditional statements these subjects found easier to use. Motivated by psycho-linguistic considerations, they attempted to compare the ease of use of the IF-THEN-ELSE construction and the IF(CONDITION)GOTO statement. Their result, based on a relatively small sample size in a carefully controlled, but artificial programming environment, was that the IF-THEN-ELSE construction was easier to use and resulted in fewer bugs, particularly with more complex problems. The paper does not mention structured programming, but the results are an initial confirmation of the concept.

Miller[16] also tested non-programmers using a highly simplified, specially constructed programming language. His subjects constructed programs for a series of simple card-sorting problems (the cards had a single name printed on them) containing conjunctive ("and") and inclusive disjunctive ("or") conditions expressed in the negative or affirmative. An example of the inclusive disjunctive form where one clause was in the affirmative and one was in the negative, was the problem of writing a program to "put a card in box three if either the name's second letter is *not* 'L' or if its last letter is 'N' . . . count the number of cards in box three using counter 1. Put the remaining cards in box 2." The results indicate that it was more difficult for the subjects to deal with the inclusive disjunctive than the conjunctive and that a negative clause made the problem still more challenging. The difficulty was measured by the time used and by the number of errors, both of which were significantly higher for "or" problems. Although this experiment does not directly shed light on language design issues, the experimental methodology is useful, and it suggests that further work with non-programmers would be very useful.

A third group[17] is comparing two proposed data retrieval sublanguages by testing programmers and non-programmers. This is the first time that a thorough and carefully controlled experiment has been performed prior to implementation.

Research in stylistic considerations

A number of texts have focussed on issues of programming style.[18,19] Although these texts offer valid practical suggestions for reducing execution time or storage utilization, they can only proffer subjective suggestions for stylistic decisions. Those stylistic issues include documentation standards, keypunching rules to increase clarity, guidelines for the selection of variable names, suggestions about programming techniques to increase the readability of programs and principles of program design.

Newsted[20] has elevated the discussion by conducting an experiment to determine the influence of comment cards in FORTRAN programs. His results indicate that on short FORTRAN programs (less than 30 lines) comments and mnemonic names may actually interfere with attempts to understand programs.

Weissman[21-23] has carried out a number of interesting experiments, concentrating on stylistic issues such as commenting, meaningful variable name selection, indentation, choice of flow of control techniques and subroutine use. Unfortunately, none of these experiments resulted in clear-cut recommendations for programmers. Weissman focuses heavily on issues of experimental design and techniques for measuring comprehension.

Research in design techniques

Some of the most provocative current debates are on the topic of program design methodologies such as modularity, step-wise refinement, and top-down design, each of which is often coupled to the structured programming concept. Although Baker[24] has reported on remarkable successes in a single project and Weissman conducted a single inconclusive experiment, no steps have been taken to confirm or refute these proposed design techniques. Personal testimonials do not suffice; replicable experimental results from a wide range of subjects are necessary.

Even the fundamental technique of flowcharting can be controversial. While some programmers reject the usefulness of flowcharting, others find it essential in planning and documenting large programs. A pilot experiment has demonstrated that for short programs, preliminary flowcharting does not simplify the programming task. A new flowcharting technique for structured programming[25] is gaining acceptance, but it has not yet been experimentally validated.

EXPERIMENTAL PARADIGMS

The fundamental difficulty in this research area is that it is so broad and so ill defined. Basic research into the way people think about programs would serve as a stepping stone to more precise experiments to resolve particular issues. But first, the underlying experimental methodologies must be developed and verified.

Problem domains

There are at least five highly interwoven tasks which unify the questions of programming language design, stylistic considerations and design techniques. In each case the goal is to facilitate the interrelated tasks of:

- learning
- program understanding
- program writing
- debugging
- modification

Preferred improvements would impact positively for each of these tasks, but it is conceivable that an improvement in one area would hinder another. For example, a complex indentation or keypunching rule might make the program easier to read and understand, but more difficult to write. While modularity may simplify debugging and modification, errors might be committed in passing parameters when the program is composed. Finally, a powerful but complex and difficult to learn concept may ease the burden of program writing.

Complications result from the fact that a technique which is beneficial in long programs may be a hindrance in short programs. Different stylistic and design rules seem appropriate for programs of differing lengths.

Another variable that must be explored is programmer ability. Useful principles for professional programmers may be too complex for novices. Preliminary results from several studies indicate that techniques appropriate for novice programmers working on short simple programs are substantially different from the techniques applied by professionals on large difficult projects.

In summary, proposed improvements must be evaluated with respect to the tasks of learning, program understanding, writing, debugging, and modification, while simultaneously considering the spectrum of programmer abilities and program complexity.

Experimental techniques

Developing suitable experimental techniques is a nontrivial task. Care must be taken to minimize the number of variables, and proper experimental controls must be established. A sufficiently large group of subjects must be secured and pre-tests or other measures obtained to ensure the homogeneity of the subjects. Replicable, objective tests must be constructed and validated. Finally, accepted statistical techniques should be applied to the data to produce results which would be acknowledged by other researchers.

Testing program understandability is the most straightforward of the tasks. After studying a program for a prescribed length of time, subjects are asked to describe in words the function of the program. Grading the responses can be difficult, but if more than one person scores each response, it should be possible to obtain reliable results. The subjects may be asked to introspect and respond as to the difficulty of the problem, say, on a scale ranging from one to ten. This simple experimental procedure may be used to compare two proposed language features or stylistic rules. Alternatively, the subjects may be told to study until they feel their comprehension is adequate, making time and correctness the measured variables.

The numerous other methods of measuring understanding include asking subjects to determine the output for a given set of inputs. Weissman utilized detailed hand simulations of execution and paragraph fill-in techniques as measures of comprehension. Simpler traces such as listing the line number executed for a given input may be useful. Two other measures are the correctness and time needed to make a specific modification or to locate a bug (which has been included in the program). Although subjective measures of difficulty have dubious value, they may be used to enhance each of the above techniques.

Finally, subjects may be asked to memorize the program. Since memorization is best accomplished by "chunking" of the program, to gain meaningful (as opposed to rote) learning, more detailed recollection suggests more thorough comprehension. This technique is described in the companion paper, entitled "Two Exploratory Experiments in Program Comprehension."[26]

To simplify grading, Newsted used multiple choice questions rather than fill-ins in testing the subject's ability to comprehend, trace execution, determine outputs, etc. Multiple choice questions ensure objective and clear-cut grading standards.

A number of recent studies have concentrated on errors in programming and in debugging techniques.[27-30] By including monitoring code in compilers researchers have been able to capture the diagnostic reports and determine which errors are most frequently made. Controlled experiments can be conducted by providing programmer subjects with listings containing one or more errors and studying how they attempt to locate the errors.

CONCLUSION

Experimental techniques are being developed to resolve the human factors issues in program development. These experimental techniques can be applied to objectively validate proposals for programming language features, stylistic guidelines, and design paradigms. Much work remains to be done to extend the scope of these experiments so that concrete recommendations can be made to professional programmers for improving the quality of their work and their productivity. Additional experimental results should enable instructors to provide better training programs.

REFERENCES

1. Dijkstra, E. W., "Programming Considered as a Human Activity," *Proc. IFIP Congress 1,* 1965, pp. 213-217.
2. Weinberg, G. M., *The Psychology of Computer Programming,* Van Nostrand Reinhold, New York, New York, 1971.
3. Baker, F. T., "Chief Programmer Teams," *IBM Systems Journal 11,* 1, 1972.
4. Mayer, R. E., *Instructional Variables in Meaningful Learning of Computer Programming.* Indiana Mathematical Psychology Program, Report No. 75-1, 1975.
5. Kreitzberg, C. and L. Swanson, "A Cognitive Model for Structuring an Introductory Programming Curriculum," *AFIPS Proc. National Computer Conference,* 1974.
6. Sammet, J. E., *Programming Languages: History and Fundamentals,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1969.
7. Higman, B., *A Comparative Study of Programming Languages,*American Elsevier Publishing Company, Inc., New York, 1967.
8. Elson, M., *Concepts of Programming Languages,* Science Research Associates, Inc., Chicago, Illinois, 1973.
9. Hoare, C. A. R., *Hints on Programming Language Design,* invited address at SIGACT/SIGPLAN Symposium on Principles of Programming Languages, Boston, Mass., October 1-3, 1973.
10. Wirth, N., *On Certain Basic Concepts of Programming Languages,* Technical Report No. CS 65, Computer Science Department, Stanford University, Stanford, California, May 1, 1967.
11. Carlson, C. R., *Programming Language Design,* Computer Sciences Department, The Technological Institute, Northwestern University, Evanston, Illinois, 1973.
12. Bohm, C. and G. Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules," *Comm. ACM 9,* May, 1966, pp. 366-371.
13. Henderson, P. and R. Snowdon, "An Experiment in Structured Programming, *BIT 12,* 1972, pp. 38-53.
14. Standish, T. A., *Observations and Hypotheses About Program Synthesis Mechanisms.* Automatic Programming Memo 9, Report No. 2780, Computer Science Division, Bolt Beranek and Newman, Cambridge, Mass., December 19, 1973.
15. Sime, M., T. Green, and D. Guest, "Psychological Evaluation of Two Conditional Constructions Used in Computer Languages," *International Journal of Man-Machine Studies,* Vol. 5, No. 1, 1973.
16. Miller, L., *Programming by Non-Programmers,* IBM Research Report RC 4280, 1973.
17. Reisner, P., R. F. Boyce, and D. D. Chamberlin, "Human Factors Evaluation of Two Data Base Query Languages: SQUARE and SEQUEL, *Proc. National Computer Conference,* AFIPS Press, Montvale, New Jersey, 1975.
18. Kreitzberg, C. B. and B. Shneiderman, *The Elements of FORTRAN Style: Techniques for Effective Programming,* Harcourt Brace Jovanovich, Inc., New York, New York, 1972.
19. Van Tassel, D. *Program Style, Design, Efficiency, Debugging, Testing,* Prentice-Hall, Englewood Cliffs, New Jersey, June, 1974.
20. Newsted, P. R., *FORTRAN Program Comprehension as a Function of Documentation,* School of Business Administration Report, The University of Wisconsin, Milwaukee, Wisconsin.
21. Weissman, L., *Psychological Complexity of Computer Programs: An Initial Experiment,* Technical Report CSRG-26, Computer Systems Research Group, University of Toronto, Toronto, Canada, 1973.
22. ——, "Psychological Complexity of Computer Programs: An Experimental Methodology," *SIGPLAN Notices 9,* 6, June, 1974.
23. ——, A *Methodology for Studying the Psychological Complexity of Computer Programs,* Ph.D. thesis, University of Toronto, 1974 (available as Technical Report, Computer Science Research Group CSRG-37).
24. Baker, F. T., "System Quality Through Structured Programming," *Proc. FJCC,* 1972, pp. 339-343.
25. Nassi, I. and B. Shneiderman, "Flowchart Techniques for Structured Programming, *SIGPLAN Notices 8,* 8, August, 1973, pp. 12-26.
26. Shneiderman, B. and Mao-Hsian Ho, *Two Exploratory Experiments in Program Comprehension,* Technical Report No. 27, Computer Science Department, Indiana University, 1974.
27. Gould, J. D. *Some Psychological Evidence on How People Debug Computer Programs,* IBM Research Report RC 4542, 1973.
28. Boies, S. J. and J. D. Gould, "Syntactic Errors in Computer Programming, *Human Factors 16,* 3, 1974, pp. 253-257.
29. Young, E. A., "Human Errors in Programming," *International Journal of Man-Machine Studies 6,* 1974, pp.361-376.
30. Gould, J. D. and P. Drongowski, "An Exploratory Study of Computer Program Debugging," *Human Factors 16,* 3, 1974, pp. 258-277.

Naive programmer problems with specification of transfer-of-control

by LANCE A. MILLER
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

INTRODUCTION

We have conducted a series of experiments concerning the programming performance of persons with no prior contact with computers other than the training received in the experimental sessions. Our objective in these experiments is to identify design principles for facilitating communication between the naive user, as a problem solver, and a computer system. We view programming as a problem-solving activity, an instance of what generally may be called "procedure specification." We believe it possible to design computers as optimal problem-solving tools only if the operating characteristics of the problem solv*ers* are known and taken into account. Consequently, we are seeking to discover the problems and processes involved in human specification of procedures, using experimental laboratory methods.

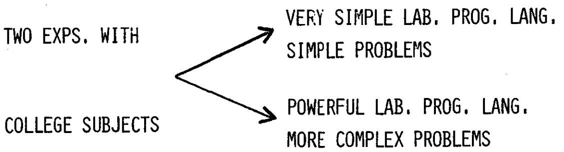
A typical research method is to create a laboratory programming language designed to investigate questions of interest, teach this language to subjects, and study performance using this language for solving a variety of problems, either using computer terminals or pencil and paper tests. In addition to this approach we have applied content analysis methods to various kinds of procedure specifications expressed in natural English, these being collected either in our laboratory or in written publications. Our objective in this latter approach is to identify normative communication modes and mechanisms which may be usable for man-machine interactions.

The presentation is organized around information and graphics charts and covers the following topics: (1) initial studies demonstrating feasibility of investigating programming in the laboratory and suggesting expression of transfer-of-control as a locus of difficulty; (2) studies comparing various means for expressing transfer-of-control; (3) results of experiments using our "procedure table"; (4) results of analysis of specifications in natural language.

Introductory and explanatory text accompanying each chart is presented below.

PROCEDURE SPECIFICATION RESEARCH

1. IDENTIFY GEN'L. PROBS.



TWO EXPS. WITH COLLEGE SUBJECTS

VERY SIMPLE LAB. PROG. LANG.
SIMPLE PROBLEMS

POWERFUL LAB. PROG. LANG.
MORE COMPLEX PROBLEMS

<u>RESULTS</u>

*** SPECIFYING TRANSFER OF CONTRAL A <u>KEY</u> DIFFICULTY (TIME ERRORS)

*** VARIETY OF CONTROL STRUCTURES PRODUCED

--- PROGRAMS W/ REDUNDANCY BETTER THAN THOSE WITHOUT

--- PROGRAMS W/ COMPLICATED FLOW OF CONTROL OFTEN IN ERROR

<u>CONCLUSIONS</u>

*** KEY PROBLEM TO FOCUS ON IS HOW BEST TO SPECIFY CONTROL

Chart I—Our first two experiments demonstrated the feasibility of studying programming in the laboratory using naive (to computing) subjects. We found that a majority of the difficulty was concerned with specifying transfer-of-control.

ALTERNATIVES FOR SPECIFYING

TRANSFER OF CONTROL

IF - THEN - ELSE (IF <u>A</u> THEN <u>Y</u> ELSE <u>Z</u>)

BRANCH - TO - LABEL (IF <u>A</u> YES: <u>A1</u> NO: <u>B2</u>

FLOW DIAGRAM

<u>OTHER</u>

DECISION TABLES

IPO & HIPO

CAUSE EFFECT GRAPHS

Chart II—There are three primary ways of expressing the set of contingencies governing appropriateness of executing a particular action—where expression is in a procedural manner.

1. GRAPHICAL --

E.G. FLOW DIAGRAMS

- - - - - - - - - - - - - - - - - - - -

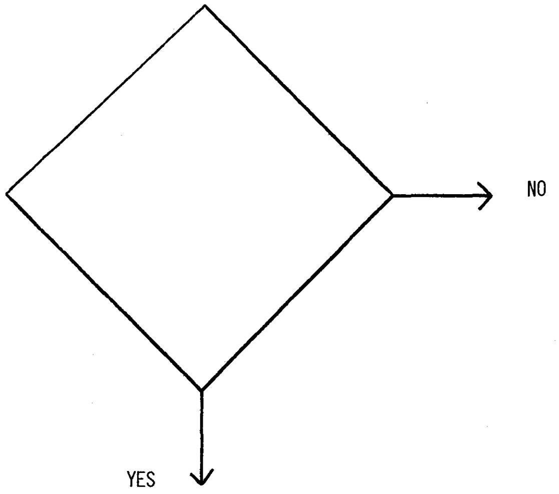


Chart III—Flow diagrams utilize two-dimensional representation with the yes/no actions to be taken depending on the truth-value of the predicate expressed as connective lines to other areas.

2. IF - THEN - ELSE

(LEADS TO "TREE" OR HIERARCHICAL STRUCTURES)

- -

IF X

THEN DO . . .

ELSE IF Y . . .

Chart IV—The "if-then-else" method involves hierarchical or nesting structures . . . (cf. also Chart VII).

3. BRANCH - TO - LABEL

(SYMBOLIC, LEADS TO COMPLEX BUT NOT EMBEDDED STRUCTURES)

- -

```
GREEN?  YES:  NEXT  NO:  B2
   .
   .
   .
B2:  DO ACTION . . .
```

Chart V—While the "branch-to-label" means permits a much wider variety of path connections (cf. also Chart VII).

EXPERIMENTAL EVALUATION IF/BRANCH/FLOW

HYPOTHESES UNDER TEST

IF VS. OTHERS

HIERARCHIC (EMBEDDED) VS. ONE-LEVEL NON-LINEAR STRUCTURES

BRANCH VS. FLOW

(1.) SYMBOLIC VS. GRAPHICAL TRANSFER

(2.) ONE VS. TWO DIMENSIONAL REPRESENTATION

PROCEDURE

GROUP TRAINING VIA BOOKLET

2 PRACTICE PROBLEMS WITH CORRECTNESS FEEDBACK

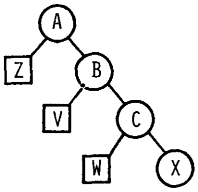
6 EXP'L. PROBS. WITH SYNTACTIC FEEDBACK ONLY

MEASURES

TRAINING - - TIME/ERRORS/COMPLETION
EXPERIM'L. - TIME/ERRORS

Chart VI—We, like Sime, Green and Guest (1973), were interested in comparing these means of expressing transfer-of-control. A comparison of the IF method vs. the others is, among other things, a comparison of planning and procedure specification using hierarchical vs. non-hierarchical structures. A comparison of FLOW to BRANCH permits evaluation of graphical vs. symbolic representation of transfer-of-control and one vs. the highly commended two-dimensional representation.

```
IF (A) THEN
    IF (B) THEN
        IF (C) THEN X
        ELSE W
    ELSE V
```



```
     IF (A)   YES A1   NO A2
A1   IF (B)   YES A3   NO A4
A3   IF (C)   YES A5   NO A6
A5   X
A2   Z
A4   V
A6   W
```

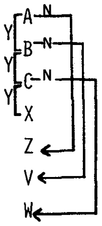


Chart VII—Use of the IF procedure restricts the variety of incorrect transfers which may occur—e.g., given the hierarchical specification of tests A, B, erroneous transfer from C to action Z instead of W cannot occur. With a BRANCH method, given appropriate means of identifying commands (command numbers or labels) transfer may occur to any other command without restriction.

EXP. EVAL. IF/BRANCH/FLOW (CNT.)

LANGUAGE FEATURES ALSO MANIPULATED

1. ABILITY TO USE LOGICAL OPERATOR "AND" "OR"

2. ABILITY TO USE NEGATION -- "NOT"

THESE WERE MANIPULATED NOT TO SEE IF DESIRABLE (THEY ARE) BUT TO FORCE DIFFERENT -- MORE COMPLICATED -- CONTROL STRUCTURES.

SUMMARY

VARIABLE	NUMBER OF CONDITIONS
PROG. MODE	3
USE OF "AND, OR"	2
USE OF "NOT"	2
PROBLEMS	6

Chart VIII—Each of the three means of expression conditionals was studied under four language-feature conditions resulting from combinations of two variables—capability to employ the negation operator "not", and capability to employ logical connectives "and" and "or". The intent of these manipulations was not to study the desirability of providing such features (they are highly desirable) but to force subjects to use various features of the conditional modes. For example, when use of the operator "not" is not permitted in the IF mode, subjects cannot simply test for "not-x" by specifying "IF not-x . . ." but they must employ the "else" aspect of the conditional—e.g., "IF X . . . ELSE K" where K represents the actions appropriate for the condition of "not-x."

EXP. EVAL. IF/BRANCH/FLOW (CONT.)

PROBLEMS: SIX PROBLEMS VARYING ON TWO DIMENSIONS
LOGICAL OPERATOR -- AND vs. OR (INCL.)
POS. VS. NEG. STATEMENT -- POS. NEW NEG. MIXED

EXAMPLE:

PUT THINGS IN BOXES INDICATED WHEN THE CONDITIONS ARE MET

BOX 4: ALL THINGS THAT ARE OLD AND NOT BROKEN
BOX 2: ALL OTHER THINGS THAT ARE OLD OR NOT WOODEN
BOX 1: EVERYTHING ELSE

Chart IX—Hierarchical multiple-action problems of the type used by Sime et al., were employed, using a fixed level of three actions and three relevant attributes. Within this framework the logical characteristics of the problems were varied on two dimensions—the logical connective, *and* vs. *or*, and the presence or absence of the operator *not*. The second action-line of the problem was always the place of variation (e.g., an affirmative conjunctive problem would read "... BOX 2: ALL OTHER THINGS THAT ARE OLD AND WOODEN ...").

EXP. EVAL. IF/BRANCH/FLOW (CONT.)

RESULTS WITH COLLEGE STUDENTS

1. TRAINING -- IF SLIGHTLY MORE DIFFICULT
2. EXP. PROBLS -- NO DIFFERENCES AMONG THREE MODES
   SIG. EFFECTS OF OTHER VARIABLES

RESULTS WITH NAVY RATINGS

1. TRAINING -- IF CONSIDERABLY MORE DIFFICULT
2. EXP. PROBS. -- IF SIG. MORE DIFFICULT, NO OTHER DIFF.

CONCLUSIONS

1. SYMBOLIC ADDRESSES AND 1-DIM. REPRESENTATION ARE O.K.
2. POSSIBLE POPULATION DIFFERENCES
3. CLOUDS OF SUSPICION FOR IF

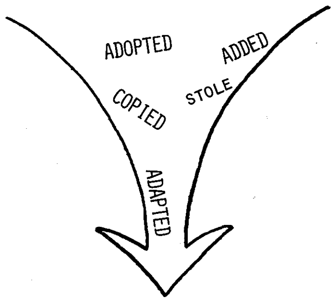
Chart X—The finding that IF was either no different from or poorer than the other modes is in contrast to Sime et al's finding of IF superiority to BRANCH. We believe that our experimental testing situation was more neutral than theirs, with automatic indentation, for example, facilitating IF performance, and forcing a jump on positive outcome for BRANCH interferring with BRANCH performance. The finding of no differences, over language conditions or populations, between FLOW and BRANCH brings into question the often expressed, but untested, claim of flow-diagram superiority over other methods.

PROCEDURE TABLE

THOUGHT ABOUT ALTERNATIVES TO

IF THEN ELSE
BRANCH TO LABEL
FLOW DIAGRAM

CONSIDERED -----------------DECISION TABLES, HIPO, SNOBOL,
NATURAL LANGUAGE CHARACTERISTICS



PROCEDURE TABLE

Chart XI—At this point we decided to test out our understanding of the processes and problems involved in programming specification of transfer-of-control by attempting to generate a method for specification which would not be subject to the problems characterizing the previously tried modes of specification, at least for the same problems. We considered a variety of possibilities, keeping in mind the insignificant BRANCH-FLOW results which suggested that one-dimensional, symbolic transfer structures did not lead to greater difficulty than FLOW representation.

PROCEDURE TABLE

LABEL	QUESTION	ACTION(S)	GO TO
A1	ANY CARD IN INPUT BOX? NO: STOP	LOOK AT NEXT CARD	
	NAME ON CARD HAS SECOND LETTER AS "NOT-L" OR LAST LETTER AS "N"	PUT CARD IN BOX #3, INCREASE COUNTER 1	A1
	------------------>	PUT CARD IN BOX #2	A1

PROBLEM: PUT A CARD IN BOX 3 IF EITHER THE NAME ON THE CARD HAS THE SECOND LETTER NOT "L" OR ELSE THE LAST LETTER IS "N" (OR BOTH).

COUNT THE NUMBER OF CARDS IN BOX 3 USING COUNTER 1.
PUT THE REMAINING CARDS IN BOX 2.

Chart XII—Our "procedure table" is in no way innovative, merely a new assembly of old ideas. If the truth value of the "question" predicate is true, transfer is made in the table to the "action" on the right in the same row, thence to the next row unless over-ridden by a specification of a command label in the "go to" area. If the predicate is false, transfer is made immediately to the next row unless over-ridden by a special "NO:" transfer within the "question" area. Provisions for transferring to other tables, etc., were worked out but not experimentally tested.

FIRST PROCEDURE TABLE EXPERIMENT

SAME PROBLEMS AS IN IF/BRANCH/FLOW EXPERIMENT

(PERMITTED USE OF "AND, OR, NOT")

TRAINING SAME METHOD -- BOOKLET -- ABOUT SAME TIME (20 MIN.)

RESULTS

NO SUBJECT HAD DIFFICULTY WITH TRAINING
------- SIGNIFICANTLY BETTER THAN FOR OTHER MODES
IN BOTH COLLEGE AND NAVY POPULATIONS

CONCLUSIONS

MAYBE WE ARE ON TO SOMETHING

BEST ALTERNATIVE SO FAR

Chart XIII—In the same manner as the experiments comparing the three conditional modes we developed training and testing procedures and materials for the Procedure Table, using the same problems. Performance was significantly better in training and in testing than with the other modes of specification suggesting that there was merit in our present tabular structuring and syntax.

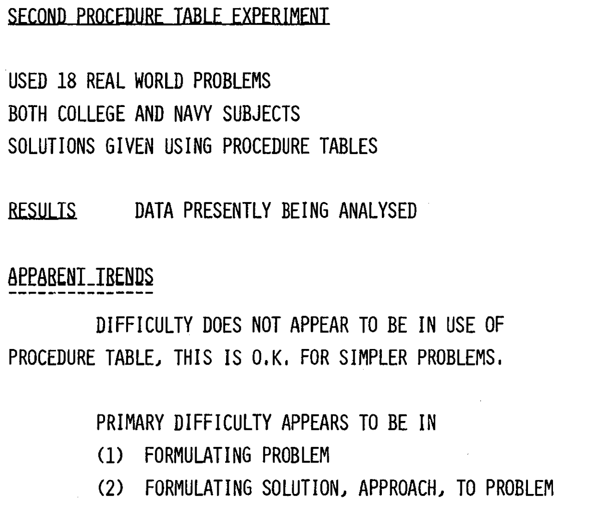
COMPLEX PROBLEMS

NOW WE ARE INTERESTED IN TAKING REAL WORLD PROBLEMS AND SEEING HOW OUR SUBJECTS FARE.

TOOK EXAMPLES FROM A NUMBER OF SOURCES

- RECIPES
- INSTRUCTION MANUALS
- REPAIR MANUALS
- MATH. PROBLEMS

Chart XIV—The problems in most of the preceding experiments were fairly simple. We now wished to see how subjects would fare in translating various complex examples of specified procedures into procedure tables.



SECOND PROCEDURE TABLE EXPERIMENT

USED 18 REAL WORLD PROBLEMS
BOTH COLLEGE AND NAVY SUBJECTS
SOLUTIONS GIVEN USING PROCEDURE TABLES

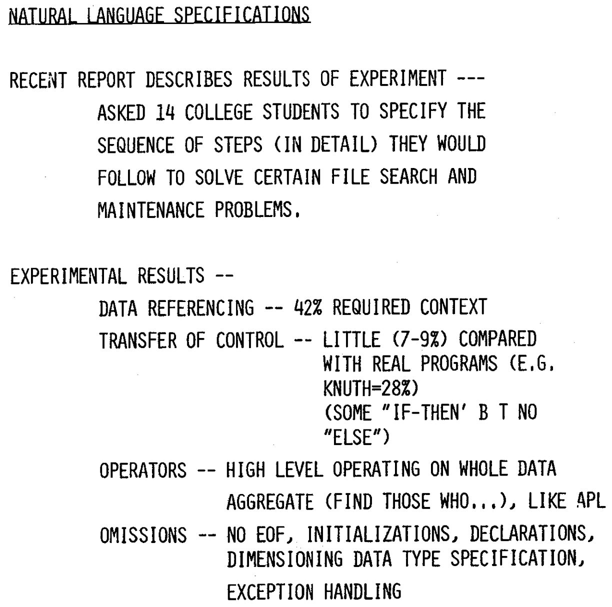
RESULTS DATA PRESENTLY BEING ANALYSED

APPARENT TRENDS

DIFFICULTY DOES NOT APPEAR TO BE IN USE OF PROCEDURE TABLE, THIS IS O.K. FOR SIMPLER PROBLEMS.

PRIMARY DIFFICULTY APPEARS TO BE IN
(1) FORMULATING PROBLEM
(2) FORMULATING SOLUTION, APPROACH, TO PROBLEM

Chart XV—While results continue to be analyzed, a key trend appears to be that incorrect specifications are due not so much to difficulty in using the procedure table but errors in understanding or resolving the ambiguity in the procedures expressed in natural language. This suggests the utility of adding pre-programming phases which facilitate planning and consideration of the solution algorithm eventually to be specified. We are currently evaluating various such planning aids.

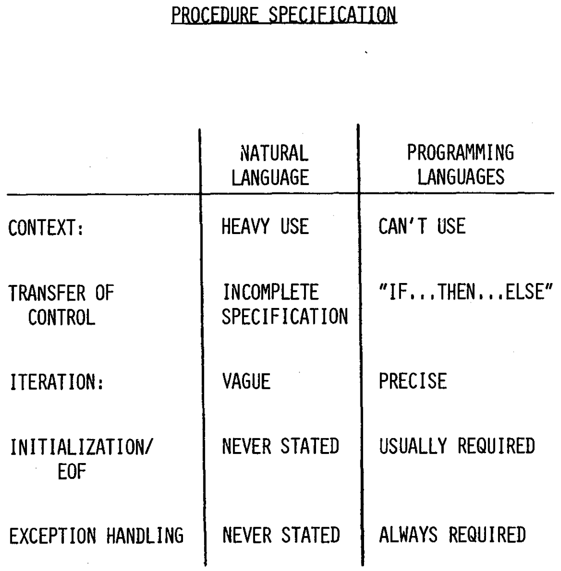


NATURAL LANGUAGE SPECIFICATIONS

RECENT REPORT DESCRIBES RESULTS OF EXPERIMENT ---
ASKED 14 COLLEGE STUDENTS TO SPECIFY THE SEQUENCE OF STEPS (IN DETAIL) THEY WOULD FOLLOW TO SOLVE CERTAIN FILE SEARCH AND MAINTENANCE PROBLEMS.

EXPERIMENTAL RESULTS --
DATA REFERENCING -- 42% REQUIRED CONTEXT
TRANSFER OF CONTROL -- LITTLE (7-9%) COMPARED WITH REAL PROGRAMS (E.G. KNUTH=28%) (SOME "IF-THEN' B T NO "ELSE")
OPERATORS -- HIGH LEVEL OPERATING ON WHOLE DATA AGGREGATE (FIND THOSE WHO...), LIKE APL
OMISSIONS -- NO EOF, INITIALIZATIONS, DECLARATIONS, DIMENSIONING DATA TYPE SPECIFICATION, EXCEPTION HANDLING

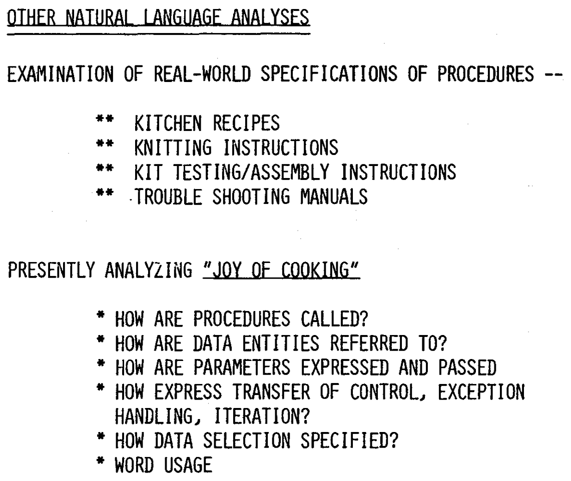
Chart XVI—In parallel with our laboratory studies of programming we have also been interested in learning about how people naturally specify procedures. Our first study involved college students writing procedures to interrogate a data base of a mythical company, and we found some interesting features . . .



PROCEDURE SPECIFICATION

	NATURAL LANGUAGE	PROGRAMMING LANGUAGES
CONTEXT:	HEAVY USE	CAN'T USE
TRANSFER OF CONTROL	INCOMPLETE SPECIFICATION	"IF...THEN...ELSE"
ITERATION:	VAGUE	PRECISE
INITIALIZATION/ EOF	NEVER STATED	USUALLY REQUIRED
EXCEPTION HANDLING	NEVER STATED	ALWAYS REQUIRED

Chart XVII—The natural language characteristics contrast strongly with features of programming languages . . .



OTHER NATURAL LANGUAGE ANALYSES

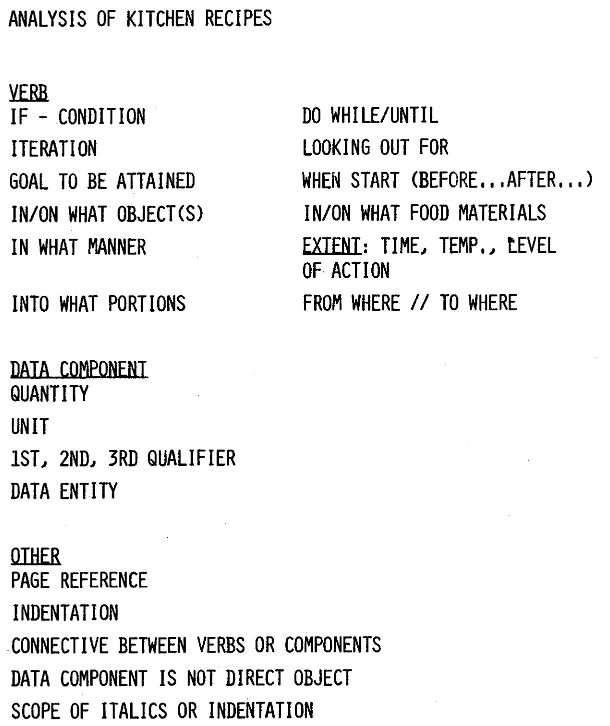
EXAMINATION OF REAL-WORLD SPECIFICATIONS OF PROCEDURES --

** KITCHEN RECIPES
** KNITTING INSTRUCTIONS
** KIT TESTING/ASSEMBLY INSTRUCTIONS
** TROUBLE SHOOTING MANUALS

PRESENTLY ANALYZING "JOY OF COOKING"

* HOW ARE PROCEDURES CALLED?
* HOW ARE DATA ENTITIES REFERRED TO?
* HOW ARE PARAMETERS EXPRESSED AND PASSED
* HOW EXPRESS TRANSFER OF CONTROL, EXCEPTION HANDLING, ITERATION?
* HOW DATA SELECTION SPECIFIED?
* WORD USAGE

Chart XVIII—In view of the interesting results obtained from this study, we felt it might be even more informative to investigate how highly stable and evolved procedures are specified—e.g., knitting instructions, kitchen recipes, kit assembly instructions. We chose kitchen recipes to begin this study (for a variety of reasons, including some important secondary considerations as evaluating output after actual execution).



ANALYSIS OF KITCHEN RECIPES

VERB

IF - CONDITION

ITERATION

GOAL TO BE ATTAINED

IN/ON WHAT OBJECT(S)

IN WHAT MANNER

INTO WHAT PORTIONS

DO WHILE/UNTIL

LOOKING OUT FOR

WHEN START (BEFORE...AFTER...)

IN/ON WHAT FOOD MATERIALS

EXTENT: TIME, TEMP., LEVEL OF ACTION

FROM WHERE // TO WHERE

DATA COMPONENT

QUANTITY

UNIT

1ST, 2ND, 3RD QUALIFIER

DATA ENTITY

OTHER

PAGE REFERENCE

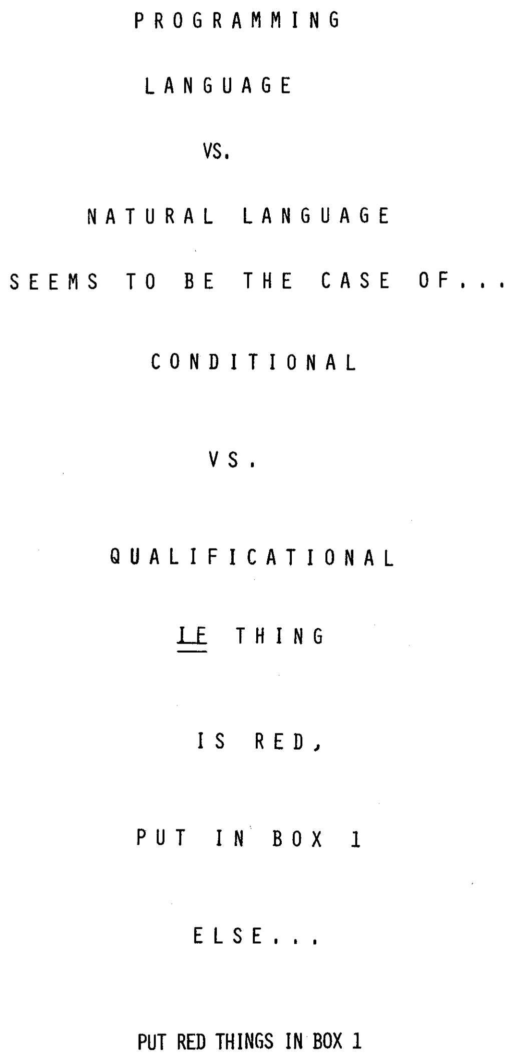
INDENTATION

CONNECTIVE BETWEEN VERBS OR COMPONENTS

DATA COMPONENT IS NOT DIRECT OBJECT

SCOPE OF ITALICS OR INDENTATION

Chart XIX--We generated and stabilized a set of semantic content categories which involves identification of two principal aspects of recipes—the action verb and the components of the recipe. Each aspect has associated with it a number of other content categories which indicate various qualifications of the principal aspects. Our analysis of verb clustering, associations between verbs and qualifiers, etc., leads us to believe that it may be quite possible to write a BNF recipe grammar. One type of analysis of immediate relevance to language design for computers is that which indicates which kind of qualifications may be omitted and under what circumstances, pointing the way for development of machine-implementable context-sensitive grammars.



PROGRAMMING

LANGUAGE

VS.

NATURAL LANGUAGE

SEEMS TO BE THE CASE OF...

CONDITIONAL

VS.

QUALIFICATIONAL

IF THING

IS RED,

PUT IN BOX 1

ELSE...

PUT RED THINGS IN BOX 1

Charts XX to XXIII—We had as one of our verb categories an *IF* classification to provide for statements of the conditions that had to be satisfied before the subsequently specified actions should be taken. This category corresponds roughly to the conditional command in programming languages, and it is of interest to note how rarely incidents of this class occurred. Most often actions or recipe components were characterized not in terms of a sequence of testing for certain attribute-values and then specifying the appropriate actions but in terms of specifying the action to be taken in certain qualified ways for certain qualified components. The qualificational method may well prove a more useful model for language design for naive users than the traditional conditional method of programming languages of the present.

REFERENCES

1. Miller, L. A., "Programming By Non-Programmers," *Journal of Man-Machine Studies,* 1974, Vol. 6, pp. 237-260. *Programming in Natural English,* IBM Technical Report No. RC5137, November 1974.
2. Sime, M. E., T. R. G. Green and D. J. Guest, "Psychological Evaluation of Two Conditional Constructions Used in Programming Languages," *International Journal of Man-Machine Studies,* 1973, 5, pp. 105-113.

Computer programming fundamentals for non-computer scientists

by DANIEL P. FREEDMAN and THOMAS PLUM
State University of New York
Binghamton, New York

BACKGROUND

Computing for Social Scientists is one of a series of introductory computer programming courses designed for specialists, this one for graduate level social science students. A wide spectrum of disciplines was represented including anthropology, sociology, psychology, history, economics, linguistics, and urban planning. The students also had a highly varied background in mathematics—from one college math course through advanced calculus; from strong statistics to no statistical background whatsoever.

Computing background was equally varied. Few of the students had any formal background in programming, but some had been users of packaged programs. Other students had worked for faculty members in a data entry capacity, and others had monitored computerized scientific apparatus.

Even though most of the students were from non-computing fields, we chose to give them a full introduction to the new approaches in computing called "structured programming." We wanted them to be aware that programs can be written with the goal of being correct as a direct result of the way they are written, rather than being thrown together in hopes of not being too hard to debug. We did not delve into the technicalities surrounding structured programming—rather we followed the non-technical approach of the introductory textbooks *Structured Programming in PL/C* (by Weinberg, Marcus, and Yasukawa), and *Structured Programming in APL* (by Geller and Freedman). This approach not only gives them a higher standard of precision in their own work, but also shows them that they can read other people's programs for correctness—freeing them, as consumers of software, from dependence on haphazard testing of the programmer's product.

This emphasis on program reading paid off in some unexpected ways. A mini-computer show was held on campus, scheduled during the class period, so there was no difficulty getting the whole class to attend the show. Both instructors attended the exhibition, to see the displays and to explain the machines to the students. Of course they were also interested in seeing how the students approached computer sales personnel. Now, for the first time, the students had a real opportunity to play the role of consumer!

A student asked the salesman to display the program for the output on the fancy CRT. The salesman tried to avoid doing this by explaining that the program was written in BASIC, a language with which the student was unfamiliar. She remained undaunted and said, if the language was called "basic", she should be able to understand it. Finally, the salesman displayed the program, and with a few words of syntactic explanation, the student was able to read the program. After examining the program for about five minutes, the student discovered a potential bug. Given a certain input, the program would blow up. The salesman said that the critical input would never be given, and the student smiled knowing full well that *she* would have given *that* input.

The *objectives and priorities* of the course were designed with the audience in mind. We knew that the majority of our students were becoming computer customers rather than computer programmers, but the emphasis was still placed on creating *usable* programs. Sheltering and reliability were the key objectives. Sheltered programs are designed with the user in mind. As far as possible, a program checks all input data for suitability. If the input is not correct, the user is informed, and the illegal input is not processed. Whenever possible the program control should remain within the program, rather than depend on the larger system for error correction or recovery.

Ease of data entry was also an important concern. The philosophy was that the user should not be exposed to all the peculiarities of the computer. The programs were designed to do formatting, both of input and output, and an output which was difficult to read was unacceptable.

The programs themselves were to be structured and readable. The programs were to be self-documenting and self-explanatory. "Let the program speak for itself."

Each program submitted had to be correct, and the correctness had to be demonstrated through a combination of structure and rigid testing. In addition to the testing, each program was examined and read by a classmate before submission. Any program which did not reach acceptable standards was returned, corrected, and resubmitted.

METHODOLOGY

The course was taught by a team of instructors, both of whom had considerable experience in computer programming and social science. Although the instructors' backgrounds did not cover the total range of student interests, most major areas were covered between the two.

The class met twice weekly for two-hour sessions. One of these sessions was a lecture presentation by one instructor; the other was a workshop. The format of the workshop varied. Some weeks the class would meet at the computing facilities and students would work on their programs and receive individual assistance. At other times, exercises would be presented, homework reviewed, and specific programs discussed. The workshops were extremely successful in teaching practical debugging technique, program reading, and error checking. They created a congenial, pleasant work atmosphere for free discussion and exploration of computer concepts.

In addition to the instructors' presentation, computer center staff provided supplementary information about our computing installation. This presentation introduced the relevant packaged programs, computer center personnel, and informed the students of the services of the computer center. Computer center involvement was motivated originally by a sad story.

One day, one of the instructors was walking across campus when he encountered a colleague from another department. The professor was unloading a full station wagon. The cargo was box upon box of punched cards. During the conversation which followed, the instructor learned that the professor had just returned from a special trip to the Midwest to pick up his data and programs. These were statistical packages which had been written for the professor at his previous university. Subsequent discussion indicated that the statistical routines were fairly standard, and were contained in the packaged commercial programs offered by our computer center. It further appeared that this professor was never told about the possibility of transferring the data and programs to tape for ease of transport.

After this sad experience, the instructor realized the importance of teaching future social science professors where to find information—who to speak with about what, and a general appreciation of the capabilities of the modern computing system.

LANGUAGE AND TEXT

It has been our experience that teaching beginners *two* languages avoids language dependence. Once a student has mastered two languages, the third or fourth is not so awesome. The vast number of computer languages combined with the high rate of academic mobility requires that computer-oriented scholars be capable of adapting to different computer languages, facilities and installations.

With these factors in mind, two radically different languages were chosen for instructional purposes—PL/C (the Cornell version of PL/I) and APL. These two languages present an excellent contrast. APL is an interactive language with good facilities for handling higher level data structures. Its workspace limitation of 32K places a limit on the size of the problem. This "natural" limitation is fortunate, because our version of APL does not lend itself to large applications.

PL/C is a subset of PL/I, enhanced with excellent error detection facilities. The usually intelligent choice of error correcting options, and meaningful error messages aids the student to overcome many of the annoying pitfalls often encountered by beginning programmers. Our experience has shown that students who have mastered PL/I have little trouble reading and mastering other popular batch languages.

The use of these languages is further enhanced by the availability of two texts, *Structured Programming in PL/C* by Weinberg, Yasukawa, and Marcus, and *Structured Programming in APL* by Geller and Freedman, specifically designed for introductory students. The elements of design, program structure, clarity, and rules of structured programming are clearly incorporated into the programming procedure. Throughout the texts, programs which are progressively constructed and modified present a clear picture of the programming process.

By reading the many programs used as illustrative material, the student makes the acquaintance of good programming practices and learns to recognize good programming techniques. This last point was extremely important for our specialized audience. As consumers of software products, professionals, regardless of discipline, should be able to evaluate relevant packages. Even if these products are free, they still require time, which is the professional's currency.

EXERCISES

Some exercises were designed to reinforce a healthy skepticism on the part of the student as a consumer. Students were asked to choose a program from the APL library which had some application in their respective fields, or was just of general interest. Students were then asked to execute these programs, as if they were naive users; that is, read and follow the directions. They were not supposed to correct or debug these supposedly tested programs but rather, report the experience. They were to evaluate the same properties we had been stressing throughout the course; i.e., reliability, clarity of documentation, ease of use, clarity of program, clarity of output, and user protection. The results were interesting to the students though not surprising to the instructors. The majority of the programs did not do what they said they did. Most of the documentation was inadequate if not incorrect, and a good percentage did not execute at all. Of those which ran, many would blow up on incorrect inputs, and most had no means of recovery. In some cases the function which was supposed to print out the instructions failed to operate. This exercise vividly impressed the students with a sense of responsibility toward the user.

Other exercises were designed to move the students into taking a familiar function or operation and writing a program to execute the algorithm. An example of the problems was assigned to read as follows: "Complete a program called ENVIRON which will print out the distributional environment for a given letter. (Assume that you already have a corpus stored in a vector called TEXT.)" By consistently moving from the familiar to the unfamiliar, a methodology of programming exercises was presented, and by the termination of the problem, all students had been exposed to many solutions to the same problem. Thus, they learned that *design* is a choice among alternatives, not the *only* solution to the problem. The importance of design was further emphasized by having students modify their code, to make them aware of the difficulty in trying to modify a poorly structured program.

Essentially, exercises were designed to acquaint the students with a wide range of computer applications. As social scientists from different disciplines and different interests within disciplines, some would be doing numeric calculations whereas others might be dealing with textual problems. We felt that regardless of their immediate interests, each group should know the computer's capability in both these areas. Even if a sociologist never dealt with textual material, he or she should be a knowledgeable consumer of software.

THE FINAL PROBLEM

Remembering that our students were specialists in the social sciences and taking a computer course as an excursion in their educational careers, we wanted the final assignment to be of relevance to their primary interests. Therefore, the last assignment was broadly expressed. The student was to choose a significant problem within his discipline which lent itself to a computer application and attempt to solve it. We did not expect the resulting analyses to be earthshattering, but we did expect them to be meaningful and interesting. We were not disappointed.

Final projects could be broken down into two major categories. Many students explored the computer center resources and found packaged programs which were useful in analyzing data they had been compiling. Some of these applications were of a standard statistical nature, where others were new applications of standard routines in an extremely creative manner. One group of archeologists applied a mapping program to archeological-geological data. They were attempting to discover trade and migration patterns based on the distribution of raw materials in certain locations through time. An examination of this new form of data representation led to the formulation of hypotheses which were subsequently tested through standard archeological fieldwork methodology. Many of the hypotheses were validated.

CONCLUSIONS

We found that teaching, rather than "exposing", introductory computer programming to noncomputer people is a time consuming task. This course, which we consider a success, required two instructors, with a heavy commitment. All student programs were carefully read and discussed by both teachers. Given a class of 26 people, this required at a minimum one full day per week. Workshops and lectures were attended by both instructors, a practice essential for continuity. Team teaching requires the full time commitment of both people rather than the half time interest of two teachers.

Teaching computer consumerism involves many of the same principles as teaching good computer programming. Be skeptical. Read programs. Ask questions, and most of all remember the words "I don't know" and "I don't understand."

BIBLIOGRAPHY

1. Yasukawa, N., R. Marcus and G. M. Weinberg, *Structured Programming Using PL/C: An Abecedarian,* New York, John Wiley & Sons, 1973.
2. Freedman, D., and D. Geller, *Structured Programming in APL,* in press, to appear Dec. 1975, Winthrop Publishers.

PART II

METHODS AND APPLICATIONS

Area Co-Director:
Bruce Wrigley
Travelers Insurance
Hartford, Connecticut

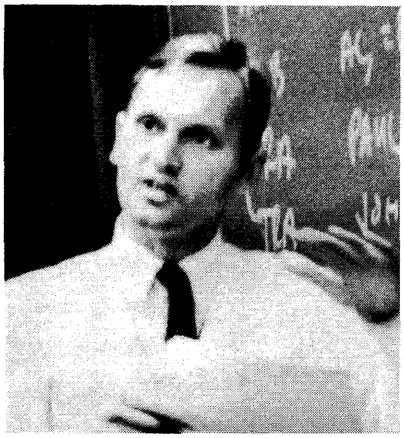
Area Co-Director:
Edward J. Palmer
Boston University
Boston, Massachusetts

User's viewpoint on EDP

Two prime concerns of the user of data processing equipment today are formalism in the administration of the DP function and the question of privacy. The former is becoming important through management pressures for engineering-like controls and procedures. Emphasis is being generated on the latter by activities of governmental bodies in proposing legislation which will control data processing practices. Both pressures will require rapid and definitive response. These concerns are addressed in a series of four panel discussions structured for the EDP user. Panelists discuss the issues and outline their solutions to some of the problems.

There have been advances in the state of the art of systems development recently, and even newer methodologies are being experimented with. In the session on "Benefits of New Programming Methodologies," the impact of these methods on the effectiveness of the organization is explored by those who have used them. One of the emerging trends in larger data processing organizations is the establishment of functional units devoted to the optimization of resource utilization. Methods, tools and results are discussed by individuals who have been deeply involved in this new activity in the session titled "Optimization of EDP Installations." There are many organizations currently involved in the selection process for data base management systems. There are very few guidelines to follow in this process and a formal technique for making such an analysis is strongly needed. A panel has been assembled for "Selection Techniques for Packaged Data Management Systems" consisting of individuals who have each used different formal methodologies resulting in the selection of different packages. The panelists will present their methodologies and argue the relative importance of various factors in such a selection. In the final session, "Issues and Answers-Data Security and Personal Privacy," some of the people who are active in the legislative process regarding privacy and security develop a picture in regard to what the user can expect in the near future, what is needed, and the implications of both to the data processing function.

No formal papers are included in this group of sessions.



Area Co-Director:
John J. Donovan
Sloan School, M.I.T.
Cambridge, Massachusetts

Area Co-Director:
Michael S. Scott Morton
Sloan School, M.I.T.
Cambridge, Massachusetts

Management and computers

As our society evolves into a more service-oriented society, the problems of management and processing information are increasing. The computer-based information system function has changed considerably in its 20-year history. Hardware and software have evolved significantly. Applications have been added in abundance. Methods of managing the technology, human resources, and system and user/computer interfaces have evolved, and most recently, the technological advances in the software for storing, retrieving, and manipulating data have made significant advances.

We have set up three interrelated sessions focusing on what computational needs user executives perceive they want, WHAT EDP professionals (computer executives) perceive managers want, and on EDP topics that managers and EDP professionals concerned with managers should be aware of.

The first is a panel session consisting of managers and executives (not EDP specialists). They will discuss the past, present, and future of EDP in their organization with emphasis on the future.

The panel consists of:

Mr. W. J. Evans
Manager, Purchasing and Administration
The Aluminum Company of America (ALCOA)
1501 Alcoa Building
Pittsburgh, Pennsylvania 15219
(412) 553-4303

Mr. H. L. Kephart
Vice President Financial Administration
Sun Oil Company of Pennsylvania
(Product Group)
1608 Walnut Street
Philadelphia, Pennsylvania 19103
(215) 985-1600

Mr. William Madden
Director of Technical Services
Bank of America Investment Management Corporation
555 California Street
San Francisco, California 94104
(415) 622-6606

Mr. Peter E. Viemeister
Vice President of Development
Grumman Corporation
Bethpage, New York 11714
(156) 575-3445

The second session is also a panel consisting of EDP specialist and computer executives. The panel consists of:

Robert B. Anderson
President
Sun Services Company
240 Radnov-Chester Road
St. Davids, Pennsylvania 19087
(215) 985-1600

Barry D. Rowe
Vice President and General Manager
Data Systems
Martin Marietta Corporation
Hampton Plaza
300 East Joppa Road
Baltimore, Maryland 21204
(301) 823-1600

Dr. Phillips Whidden
Manager
ALCOA Management Information Services Division
1501 Alcoa Building
Pittsburgh, Pennsylvania 15219
(412) 553-3264

They will focus on their views of EDP in their organization and in general.

The third session is a paper session focusing on three EDP topics that managers and EDP specialists should be aware of. The first is a cost analysis of interactive computing in organizations. The second is a new technology that is particularly helpful in building management information systems. The third is an approach to analyzing EDP requirements.

Practical guidelines for EDP long-range planning

by JOHN V. SODEN and GEORGE M. CRANDELL, JR.
McKinsey & Company, Inc.
Los Angeles, California

INTRODUCTION

This paper summarizes the approaches some major organizations are taking toward EDP long-range planning (LRP) and presents what we believe are useful and practical guidelines for others involved in, or contemplating, such an activity. The information presented here was gathered during an April 1974 working conference on long-range planning for information systems cosponsored by McKinsey & Company, Inc., and the University of California at Los Angeles Graduate School of Management.

Some 20 major public and private sector EDP executives attended the invitational conference, and participated in two days of discussions regarding various aspects of their LRP experience. These executives also participated in a detailed preconference survey focusing on the objectives, development process, and end products of their individual planning efforts.

The paper is divided into three sections which discuss:

1. The approaches to EDP planning utilized by the conference participants, including their planning objectives and final LRP documentation.
2. The planning processes used to develop an EDP strategy statement and LRP.
3. Practical planning guidelines that should be considered by the EDP executive in undertaking and carrying out a planning activity.

For the purposes of this paper, it will be assumed that the EDP organization, by which the planning is being done, is responsible for computer operations, systems analysis, and system development activities.

APPROACHES TO PLANNING

The approach to EDP MIS planning adopted by an organization reflects many things, including the charter, credibility, and capability of the organization itself; its industry characteristics; and the maturity, sophistication, and enthusiasm of the company and individual "users." In order to obtain a good cross section of various approaches to planning, conference participants were chosen to represent the aerospace, airline, business equipment, chemical, consumer goods, insurance, medical services, petroleum, and utility industries as well as government and education at the local, state, and Federal levels. The "average" participant was from an organization that had annual revenues or a total budget greater than $1 billion, an annual EDP budget of over $15 million, formal capital allocation and budgeting procedures, and a formal EDP long-range plan for more than three years. They, therefore, represented a cross section of relatively large, mature EDP organizations experienced in EDP planning. Hence, it can be expected that they were familiar with the literature on this subject represented by the References 1 through 8.

Planning objectives

These organizations tended to have five primary objectives for their LRP effort. Ranked in order of importance, they were to:

1. Increase communications and cooperation between the EDP group, its users, and top management.
2. Improve EDP requirements forecasting, allocation of EDP resources, and short-term decision-making within the EDP group.
3. Identify opportunities for improvement and cost reduction within the EDP group.
4. Identify new and/or higher payout computer applications and cancel marginal development efforts.
5. Gain a better understanding of, and increase EDP's visibility within, the overall organization.

Only half of the participants indicated that they attained these objectives above their initial expectations. This, we feel, was caused by several basic shortfalls in, and misunderstanding of, the planning process, as discussed more fully in a subsequent section.

LRP documentation

The vast majority of conference participants agree that something called an LRP document should contain the following:

- Statement of objectives for the EDP organization.
- Projection of future EDP and user environments.
- Application development priorities and timetable (*including specific project descriptions with cost, benefit, return on investment, and risk estimates*).
- Budget, hardware, personnel, facilities, education, and organization schedules.
- Implementation plan.

There was a difference of opinion, however, over whether certain additional items of information should be included in the document. This information consisted of:

- Summary of EDP organization strengths and weaknesses.
- Alternative strategy definitions and evaluations.
- Projection of host organization's future industry environment.

The objections to including a summary of EDP group strengths and weaknesses in the LRP document were understandable; however, an honest introspective analysis of the current status of the EDP group is necessary, both to identify areas in which improvement is required and to assess the resources available for undertaking any future plans. Regarding alternative strategies, most organizations implicitly considered them, but felt that is was not worth the time to formally document them. Finally, a projection of the host organization's future industry environment was usually left out because the company itself had not developed or documented one—the EDP group either did not desire to or did not feel capable of doing so. This is somewhat paradoxical in that one of the planning objectives was to gain a better understanding of the overall organization—it may explain why that objective was not achieved to a higher degree.

One of the most telling findings regarding the LRP documentation was the fact that, of the participants who agreed that they should include certain information, over half of them failed to do so. This information consisted of return on investment or risk estimates for potential projects, alternative strategy definitions and evaluations, and summaries of the EDP organization's strengths and weaknesses. The failure of these organizations to include items they deemed necessary indicates that even in these relatively large and mature EDP organizations, the gap between recognized standards of planning and actual practice is relatively large. As the next section discusses, this gap is, in part, a result of varying interpretations of what actually constitutes long-range planning.

PLANNING PROCESSES

The planning process is the sum of the individual activities involved in developing the LRP. It can take an endless number of forms depending upon the desired comprehensiveness of the end product and the resources available for its development. Some factors that affect the choice of an LRP process are:

- The characteristics of the overall organization—e.g., contingency planning could be very important in a public sector organization subject to rapid changes in administration and direction, while its use might be minimal in a stable manufacturing company.
- The objectives of the overall organization regarding its use of computing resources—i.e., is EDP to be a reactive support function or an assertive agent for improving operations and procedures throughout the company?
- The specific role of the particular EDP organization—e.g., a corporate EDP organization having advisory, but not operating, responsibilities might stress policy planning as opposed to operational planning.
- The managerial sophistication of the overall company and individual "user" executives—e.g., a company without established capital allocation criteria may have difficulty in agreeing on methods for ranking system development projects based on financial attractiveness.
- The relative degree of maturity and sophistication of the EDP function—i.e., an EDP organization struggling to manage computer operations on a timely, accurate, and cost-effective basis should likely concentrate on this area and not be overly concerned with planning to meet the further information needs of users.

A distinction should be made between the development of an *EDP strategy statement* and an *EDP plan*. The two are closely interrelated in that the strategy statement defines the objectives and policies of the EDP organization, while the plan lays out the actions to be taken to achieve the desired objectives. Few participants actually made this distinction and explicitly created an EDP strategy statement as part of their planning process. This appears to be one of the major factors that inhibited achievement of planning objectives.

EDP strategy statement

The strategy statement is a key communication vehicle between EDP and top management. Through it, top management can participate in directing the EDP effort such that it best suits the present and expected needs of the company. The development of an EDP strategy is also a means for the EDP executive to improve his relationship with top management and establish himself as a member of the management team.

One form of the EDP strategy statement development process is depicted in Figure 1. It involves two stages: (1) EDP management develops and presents to top management an environmental assessment, recommended objectives, and proposed policies; and (2) top management re-

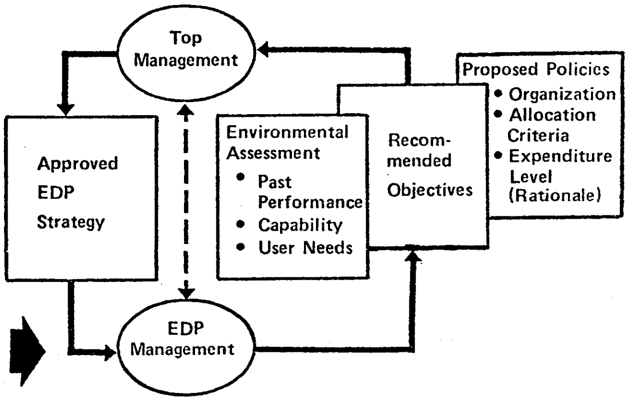


Figure 1—General strategy development process

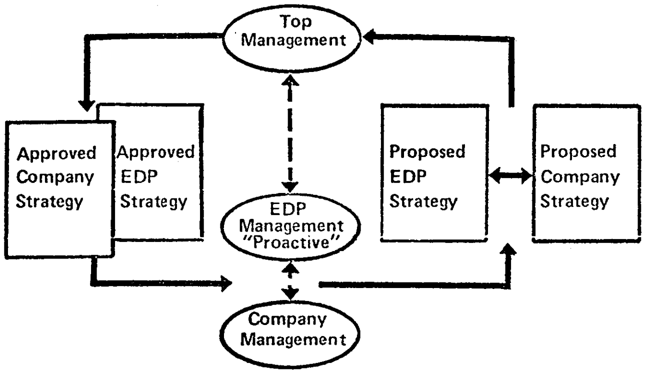

Top Management
Approved Company Strategy
Approved EDP Strategy
EDP Management "Proactive"
Proposed EDP Strategy
Proposed Company Strategy
Company Management

Figure 3—"Proactive" strategy development process

views and modifies the proposals in accord with its views or desires, thereby producing an approved EDP strategy. Through this top management participation, EDP plans which are later developed will be relatively integrated and aligned with those of the overall company.

The inputs provided by EDP management—i.e., the environmental assessment, recommended objectives, and proposed policies—indicate where the EDP group is, where it should be going, and what some of the guidelines for getting there should be. The assessment, then, evaluates the past performance of the EDP group, its current strengths and weaknesses, and the needs of its users that it is or should be satisfying. Recommended objectives for the EDP organization are generally directed at overcoming present shortcomings and preparing it to satisfy future demands. Proposed policies might cover such topics as the organizational approach to computing (e.g., centralized versus decentralized); the criteria for allocating computer-related resources (e.g., application steering committee versus first-come-first-served); and the expenditure level for EDP services, including an explanation of what services would be provided and why that level of service is appropriate.

Within the general strategy development framework there are (at least) two modes in which EDP can operate: "reactive" and "proactive." The "reactive" mode is depicted in Figure 2. In it, top management develops an overall company-wide strategy and EDP management "reacts" to it by developing a proposed EDP strategy that supports it. In this manner, the EDP strategy does not impact the development of the company strategy. In the "proactive" mode, the EDP and company strategy are developed simultaneously, as shown in Figure 3. In this arrangement, EDP can influence the company's strategy formulation such that it takes the best possible advantage of the computing capabilities and available resources.

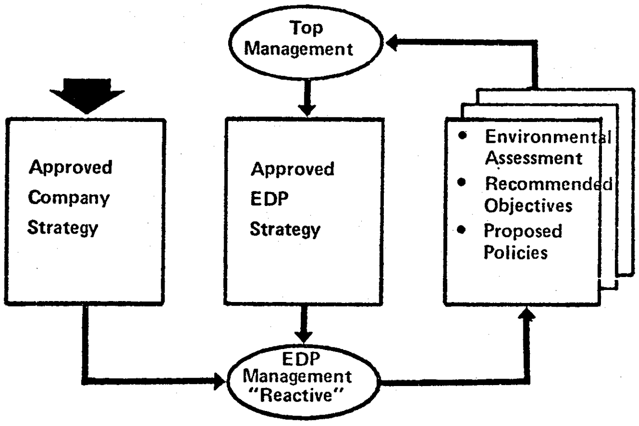


Figure 2—"Reactive" strategy development process

In both these modes the choice of appropriate strategic objectives for EDP is key since they set the tone for the long-range planning effort and the pattern for EDP's future role and development. What the appropriate objectives are for a given organization depends on its current status and anticipated future role, and these will change as the EDP group develops. A typical EDP organization might, over time, pass through the following objectives:

- Build a sound foundation for managing the EDP activity day to day.
- Cultivate a core group of appreciative users.
- Filter requests of core users to enhance the return on investment (ROI) of EDP.
- Expand the user group to further enhance the EDP ROI.
- Identify and prepare for user needs likely to arise in the future.
- Influence the future development of the company to make the best possible use of EDP.

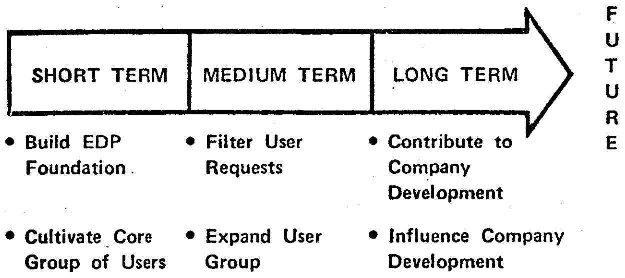


Figure 4—Strategic objectives

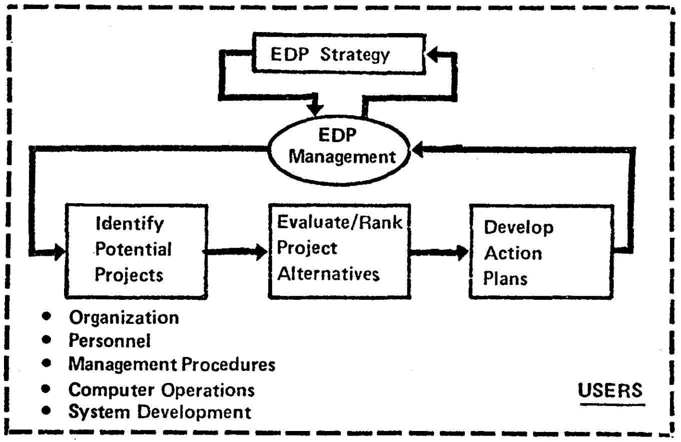


Figure 5—EDP plan development process

The above objectives can, for the purposes of convenience, be divided into three groups: short term, medium term, and long term, as shown in Figure 4. These groupings are based on the time horizon for planning implied by each objective and indicate the type of plan appropriate for achieving it.

EDP plans

As mentioned previously, the EDP plan lays out the actions to be taken to achieve strategic objectives. Thus, the plan is the end result of identifying, evaluating, and ranking alternative projects. The process of developing the EDP plan is diagrammed in Figure 5. The term "projects" in this instance is meant to include not only systems development, but all aspects of EDP activities and management, e.g., organizational modification, personnel development, improvement of management procedures, and improvement of operations. Note that the entire planning process takes place within the organizational framework of the users which it is EDP's main function to serve.

EDP plans can be loosely categorized as short term, medium term, and long term, corresponding to the major objectives they are designed to achieve (see Figure 4).

Short-term plans stress the achievement of precise quantitative objectives during a one-year period. These objectives usually relate to ways of measurably improving such areas as production service levels and costs per transaction, effectiveness and efficiency of system development efforts, and maintenance request throughput. The short-term plan generally takes the form of a statement or list of improvement targets, a schedule with milestones for reviewing progress, and an operating budget.

Medium-term plans generally seek to maximize the contribution of the EDP effort in meeting today's needs using a multiyear planning horizon. This usually involves a study to determine current users needs. A number of approaches to this may be taken, including the simple listing of all previously proposed system development projects, full-scale documentation of the types of information users throughout the company would like to have available (perhaps embodied in a company MIS architecture diagram), and/or an identification and analysis of key functional areas in the company for which data processing applications might be of high value. This last approach, often referred to as "top down" business analysis, involves an examination of the company's financial and operating data to identify those areas in which the use of EDP could have the greatest impact on the company's current and future success. Mastery of this approach poses a significant challenge to the EDP executive in that it requires him to become a businessman and develop a broad-scale understanding of the economic leverage points of the company which he serves.

Once projects have been identified, they are then ranked and a course of action developed using those policies developed for allocating computer-related resources (as well as internal EDP organization technological policies). The medium-term plan usually consists of a multiyear system development schedule and budget extrapolation to support it.

Long-term plans seek to contribute to, or influence, the future development of the company. "Contribute to" (reactive) plans involve aligning EDP activities with the long-term plans of the overall company. This might involve such things as developing expertise in new areas anticipated to be of use in the future or gradually modifying the organization structure to be better prepared to provide future services. "Influence" (proactive) plans involve active participation of EDP in the long-term planning of the overall company, e.g., making organizational recommendations based on information flow.

Naturally, planning processes cannot all be categorized in such a black and white manner; for instance, a medium-term systems development plan derived from a truly long-term systems development plan might well focus on future rather than current needs.

Most importantly, however, the use of each successively longer ranged plan should be mastered before proceeding to develop the next. This is because the successful implementation of each longer range plan is dependent upon the shorter range plans developed to support it. For example, regardless of how good a given long-range plan is, it will likely lead to disappointing results if adequate short- and medium-range plans have not been mastered for putting it into action.

Note also that the need for top management involvement is not an exclusive aspect of long-term planning. EDP objectives and policies should be established with top management's participation, even if a short- or medium-term plan is most appropriate for a given EDP organization. And these objectives and policies should be carefully set in light of an objective assessment of the EDP organization's capabilities and weaknesses.

PRACTICAL PLANNING GUIDELINES

During the conference, a great deal of discussion centered around the pitfalls involved in developing a planning approach and carrying out the planning process. Based on those discussions, it would appear that the EDP executive undertaking a planning activity should consider the following guidelines within the framework of his own unique environment.

- Recognize the growing need for formal long-range EDP planning as computer systems become more complex, cost more, require longer to develop, involve multiple functions or departments, and become increasingly important to the success of the company.
- Admit that a set of extrapolated budget, personnel, and hardware schedules routinely submitted to the corporate planners does not constitute an EDP long-range plan.
- Recognize the importance of communication with and support of top management for the planning effort—gain their agreement on a simple set of EDP policies and objectives at the outset and keep them appraised of progress.
- Define specifically the objectives of the planning effort at the outset and "plan the plan" around achieving these—do not attempt the great leap forward.
- Select a planning process suited to the practicalities of your situation—e.g., time and resources, (if possible, develop a conceptual model of the planning process, along with a well-defined set of desired end products)—avoid trying to do everything at once.
- Make the plan one for company/user use of computer systems not one just for the EDP department—involve users in its development and make its language and structure as user-oriented as possible.

Most importantly, remember that helpful as formal long-range planning can be, it cannot replace the political sensitivity, entrepreneurship, conceptual contribution, and basic business leadership required of the successful EDP executive.

REFERENCES

1. Brown, W. F., R. E. Biband, G. L. Hodgkins, "Planning For The Future Computer Complex," *Computer Decisions,* January, 1973, pp. 30-35.
2. Dearden, J., F. W. McFarlan, W. M. Zani, *Managing Computer-Based Information Systems.* (R. D. Irwin: Homewood, 1971), Chapters 2 and 7.
3. Harvath, C. and B. Bridging Chilcoat, *The Systems Expectations Gap,* an AMA management briefing, 1973.
4. Kriebel, C. H., "The Strategic Dimension of Computer Systems Planning," *Long Range Planning,* September 1968, pp. 7-12.
5. McFarlan, F. W., "Problems In Planning The Information System," *Harvard Business Review,* March/April, 1971, pp. 75-89.
6. Neuschel, R. F., "You, Your EDP Plan, And Top Management," keynote address at the Third Annual Conference of PIMA on April 13-14, 1971.
7. Schwartz, M. H., "MIS Planning," *Datamation,* September 1, 1970, pp. 28-31.
8. Steiner, G. A., *Top Management Planning,* (McMillan, 1969).

An application of a generalized management information system to energy policy and decision making—The user's view

by JOHN J. DONOVAN, LOUIS M. GUTENTAG, STUART E. MADNICK and GRANT N. SMITH
Massachusetts Institute of Technology
Cambridge, Massachusetts

Motivations for flexible systems for energy

As a result of recent disruption in the world petroleum market and rapid price increases, the United States is in the process of developing energy policies that will lead to a greater degree of energy self-sufficiency, and to a reduced level of vulnerability to interruption of supply from abroad.

New England is particularly susceptible to disruption in energy supplies, as we are "at the end of the pipeline."

One advantage of the market system is that public officials can get by without knowing much about the details of the operation of most sectors of the country. Many goods and services are produced, allocated over space and time, and delivered to consumers without government intervention and with no need for a public record of how things are done. When events occur that call for government efforts to influence markets, however, a dearth of public information can be a crucial barrier to effective policymaking.

The need for information, hence an information management system, is obvious in a crisis situation. However, there also exists a need for energy information in a non-crisis situation to aid in a wide set of tasks:

- studies of the economic impact of various events in the energy sector
- studies of the location of major energy facilities (ports, refineries, etc.)
- development of early warning indicators of problems in regional energy supply
- provision of information for special studies of environmental impacts, conservation efforts, price trends, etc.

Our objective is:

To establish a facility (for storing and validating data, retrieving data, interpreting and analyzing data, and constructing and applying models using those data), which will facilitate New England energy policy analysis and decisions.

A system to support the objectives outlined would not be adequately represented by, for example, an accounting system. The accounting system operates on a well-defined set of data in a well-defined way. Neither data nor operations are subject to rapid alternations. Furthermore, the data is relatively "clean", i.e., from consistent, high-quality sources.

For the purposes of the energy information system, the problem area being addressed is not constant. It changes when changes in perception arise, which may be for any number of reasons. This has the effect of changing both the data required and the format of data required far more rapidly than the reporting and data gathering procedures can be altered to reflect the new needs. As such, the already inaccurate data become rapidly less suited to the task at hand.

Furthermore, with change occurring so frequently, it is imperative that the system can be modified to meet the change without incurring prohibitive expenses.

While these requirements are certainly true in the energy information system, they are by no means unique to it. Our approach has thus been to meet the needs of the energy system without actually implementing an energy-specific system. Rather, we have concentrated on constructing a **G**eneralized **M**anagement **I**nformation **S**ystem (GMIS) that meets requirements of extreme flexibility, acceptable costs, and simultaneously serving a diverse user group. This paper is addressed to a particular instance of the GMIS, namely, its use in the **N**ew **E**ngland **E**nergy **M**anagement **I**nformation **S**ystem (NEEMIS), and more specifically, to the user view of the system rather than the implementation.

DESCRIPTION OF NEEMIS

Keeping in mind the ultimate purpose of NEEMIS—to provide a facility to aid public policymakers in energy decisions in New England—we recognize several classes of users of the NEEMIS facility. In this section we shall briefly explain what facilities each class of user will have. The precise syntax of intermediate languages and implementation details are described elsewhere.[1]

INTERFACE FACILITY USER CATEGORY	PREPARED PACKAGES	NEEMIS INTERACTIVE QUERY FACILITY	NEEMIS HIGH LEVEL QUERY LANGUAGE	MODELING FACILITY	RELATIONAL QUERY LANGUAGE (DML)	DATA DEFINITION FACILITY (DDL)	RELATIONAL OPERATORS & PL/1 FACILITY
NON-TECHNICAL (No computer training)	X	X	X				
WELL TRAINED	X	X	X	X	X		
RESEARCHER	X	X	X	X	X	X	
SYSTEMS ANALYST & PROGRAMMERS	X	X	X	X	X	X	X

Figure 1—Project *NEEMIS*—Functions available in each interface facility

In the NEEMIS facility, we wish to give users increasingly more powerful tools. Figure 1 depicts 4 classes of users as factors:

- Non-technical—e.g., a state energy officer. His objective is to get answers to questions and report.
- Well-trained—e.g., a specialist within a state energy office who has been trained in the use of the system.
- Researcher—e.g., an economist with some computer background who wishes to build a model for a special study.
- Systems analyst/programmer—e.g., a computer professional. He may wish to add a new table to the system or change the protection rights on an existing data series.

Looking across the table in Figure 1, we see the tools available to users of NEEMIS. Although all levels and facilities of the system are available to all users, it is unlikely that users will venture outside of those tools designated (by "X"). Increased sophistication on the part of any one user will, of course, qualify him/her for a different category.

The tools of the system have been designed in such a way that the interests of the various user groups are met. Let us proceed to briefly describe the facilities at each level.

Relational Operator and PL/1 Facility

At this level, the user sees all data as being stored in *relations.** This includes not only regular entered data, but all system data, all access control data, etc. The user at this level has at his command thirteen set-oriented relational operators that are used to perform *all* operations on *all* data. It is important to note that user data, system data, access control data, etc., are all accessed in a consistent manner via these thirteen operators that are based on the relational model of data,[2,3] which have their roots in logical systems and predicate calculus.[4-7] The operators available in NEEMIS are described in detail in Reference 1.

Since these operators appear as PL/1 subroutine calls within NEEMIS, the user at this level also enjoys all the power of PL/1.

Notice that both PL/1 and relational operators require precise use and exhibit low tolerance for error.

Data definition facility

A user at this level has facilities to specify and create relations. We call this facility the Data Definition Language (DDL). The DDL will accept a data specification and will produce an appropriate relational data base, which is then incorporated into the system. The DDL also provides a facility for loading bulk data into the newly constructed relational system from punched cards, magnetic tapes, or magnetic disk files.

In the establishment of a new relation, the system tables are modified to include data about this new relation, as well as provision for specification of access control, etc.

Also available at this level is on-line help with commands, and extensive diagnostics.

An example of the use of the DDL facility follows. ("Domain" means a column of the "relation," or matrix.)

* For our purposes, can be thought of as matrix of values; each column a domain, each row an entry. See Reference 2 for more details.

Example:

With most information management systems, the design of the system—that is, the design of the data base—is a vital step in the operation. If done incorrectly, it is often impossible, and usually extremely costly in dollars and man years to restructure the data base to more ably suit the needs.

Not so with NEEMIS. In fact, during the summer we experimented with three different designs in the course of a single month. The DDL permits specification of the data base on-line, and extremely rapidly. A sample session might be:

Example:

```
system:  ENTER COMMAND:
user:    define domains
system:  .                      (.="ready for input")
user:    name character, soc_sec_# numeric (9),
user:    supplier choice (gulf, exxon, mobil),
user:    address character;
system:  ENTER COMMAND
user:    create relation
system:  .
user:    employee (name, soc_sec_#, address)
           (primary key: soc_sec_#),
         fuel_data (soc_sec_#, name, supplier)
           (primary key: soc_sec_#, required:
           supplier);
system:  RELATIONS DEFINED
         ENTER COMMAND
user:    define synonym: soc_sec_#='ss';
system:  SYNONYM ENTERED
         ENTER COMMAND
user:    stop.
```

This session would establish the two relations, and permit data to be entered immediately.

Query facility

At this level a user can specify queries of data stored in relations. The user uses a rigid syntax for his queries that we sometimes call "cryptic" English. More specifically, we call this facility a *Data Manipulation Language* (DML).

An internal document describes a complete DDL and DML that has been specified at M.I.T.[18] Other attempts at implementing a query facility based on the relational model include: MACAIMS,[8] SEQUEL,[9] COLARD,[10] RIL,[11] and M.I.T.'s RDMS.

This facility is available for querying relations established via the DDL or possibly the relational operator facility (see earlier sections of this paper).

The commands, although conforming to a rigid syntax, employ English-like keywords, are quite easy to learn and readily readable. Once again, all data, including system data, are accessed in a consistent manner; and access control specification is an integral part of DML.

Let us give two examples here of our DML query commands.

We assume that the following four tables have been created using the DDL. The first table is named 'terminal' and it has six columns: terminal id, name, etc.

TERMINAL (TERMINALID, NAME, CITY, STATE,
  ZIP CODE, AFFILIATION)
SUPPLY CAPACITY (TERMINALID, FUELTYPE,
  FUELAMI, DATA)
SUPPLIER (SUPPLIERNO, NAME, VOLUME,
  FUELTYPE, DISTNO)
DISTRIBUTORS (DISTNO, NAME, ADDRESS,
  CITY, STATE, INVENTORY, FUELTYPE)

The following are sample queries against a data base that contains the above tables:

Question 1

```
DISPLAY NAME, AFFILIATION, CITY
  FOR STATE='MASSACHUSETTS'
```

This question causes the listing of the name, affiliation and city of all terminals in the state of Massachusetts.

Question 2

```
DISPLAY NAME FOR FUELAMT>1000,
FUELTYPE='GASOLINE', CITY='LYNN'
```

This lists the names of all terminals in Lynn that have over 1000 gallons of gasoline capacity.

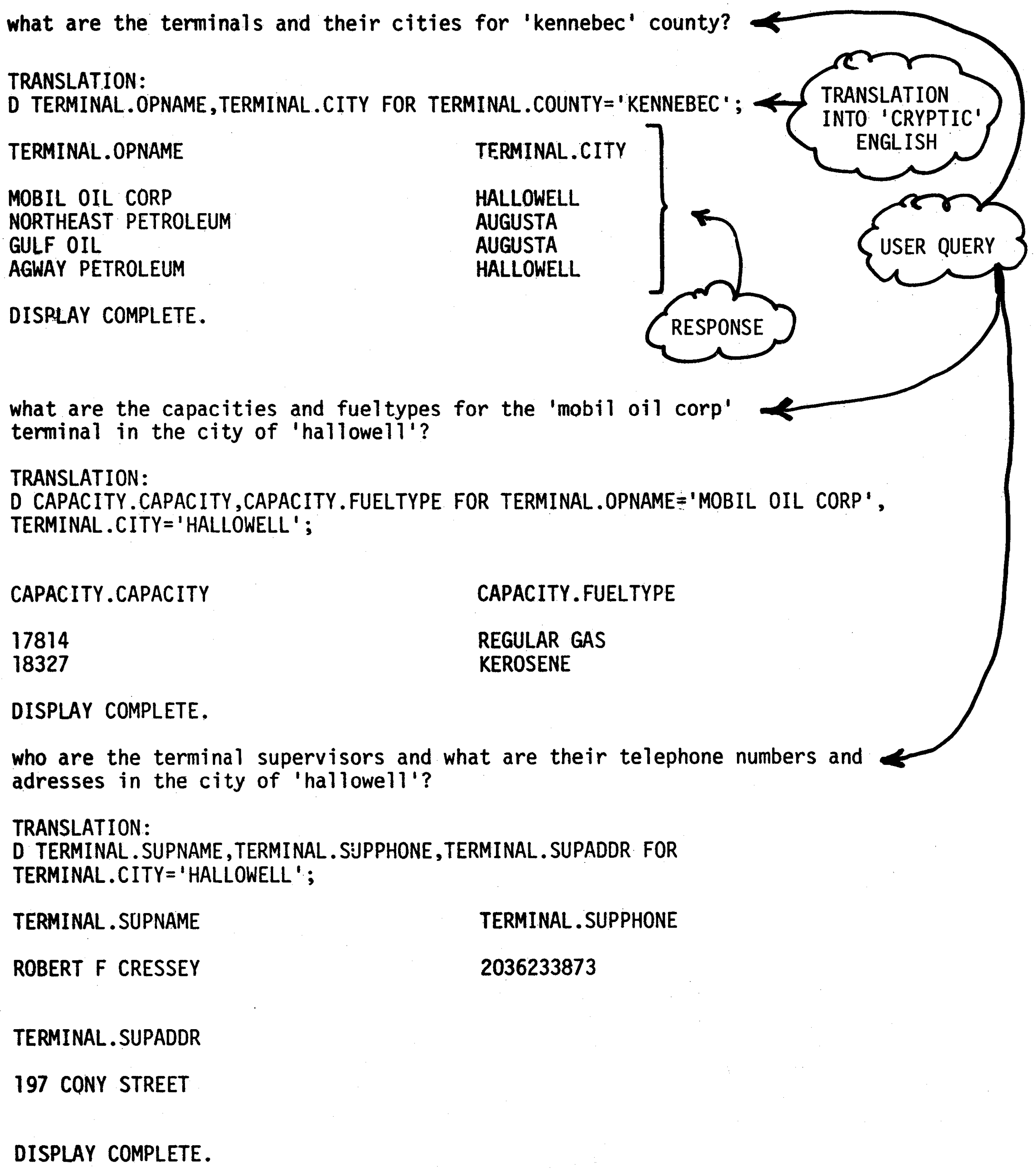
The display command is but one of several available. All commands employ consistent syntactic constructs and are equally readable.

There is, again, extensive on-line help with commands available, as well as explanatory diagnostics. No high-level user should have to see "protection exception at location OFE1A3"!

The modeling facility

A user of this facility may construct and activate a model interactively via provision of a set of functions called from APL. These functions include regression routines, plotting routines, time series modeling routines, etc., in addition to the standard APL facilities. The language used for modeling is a superset of APL—i.e., APL with additional facilities. The data that the model uses may be retrieved directly from stored in the relations (see previous section).

This APL-oriented modeling facility is the current standard. Inclusion of additional or different modeling languages, however, poses little problem (see later section).



```
what are the terminals and their cities for 'kennebec' county?

TRANSLATION:
D TERMINAL.OPNAME,TERMINAL.CITY FOR TERMINAL.COUNTY='KENNEBEC';

TERMINAL.OPNAME                       TERMINAL.CITY

MOBIL OIL CORP                        HALLOWELL
NORTHEAST PETROLEUM                   AUGUSTA
GULF OIL                              AUGUSTA
AGWAY PETROLEUM                       HALLOWELL

DISPLAY COMPLETE.


what are the capacities and fueltypes for the 'mobil oil corp'
terminal in the city of 'hallowell'?

TRANSLATION:
D CAPACITY.CAPACITY,CAPACITY.FUELTYPE FOR TERMINAL.OPNAME='MOBIL OIL CORP',
TERMINAL.CITY='HALLOWELL';


CAPACITY.CAPACITY                     CAPACITY.FUELTYPE

17814                                 REGULAR GAS
18327                                 KEROSENE

DISPLAY COMPLETE.

who are the terminal supervisors and what are their telephone numbers and
adresses in the city of 'hallowell'?

TRANSLATION:
D TERMINAL.SUPNAME,TERMINAL.SUPPHONE,TERMINAL.SUPADDR FOR
TERMINAL.CITY='HALLOWELL';

TERMINAL.SUPNAME                      TERMINAL.SUPPHONE

ROBERT F CRESSEY                      2036233873


TERMINAL.SUPADDR

197 CONY STREET


DISPLAY COMPLETE.
```

Figure 2—Example of computer dialogue

NEEMIS high-level query facility

Figure 2 shows an example of the type of query that can be used at this level. For purposes of illustration, we have shown how the requests are translated into DML and passed to that level for further handling. ('D' is an abbreviation for "DISPLAY".)

NEEMIS interactive query facility

The user of this facility simply points to a question category he wants answered on a CRT using a "light pen". If the question needs further specification, the system will flash subsequent choices on the scope, and the user will point to the choice that clarifies his query.

Prepared packages

Users of this facility will request standard reports or invoke common models, for example, a monthly forecasting model. All the user at this level needs to know is the *name* of the report or model. The system will take care of retrieving the requisite data and invoking the appropriate facility to generate a report or run a model.

NOTES ON IMPLEMENTATION

The purpose of this paper was primarily to describe the hierarchy of user facilities in NEEMIS as opposed to a description of the implementation of the GMIS. However, there are a number of interesting implementation-related points that bear mentioning.

Extensions of the relational model

Just as the user-view of NEEMIS described levels of differing power and flexibility, so the actual implementation of the system was carried out. Software developed for the GMIS has been implemented as a multi-level hierarchy in which each level employs only those facilities implemented in the levels below it. Early explanations and applications of this approach may be found in References 12, 13, and 14.

The GMIS in which NEEMIS is built has paid extensive heed to security of data. Some nineteen types of access have been identified and any owner of data may authorize any user to access those data in any or all of those nineteen ways. The default authority is NO access, rather than the usual approach that allows full access unless otherwise specified. These security specifications are made via facilities in the DML directly.

The relation used to store access control information, as well as all other system relations and descriptors are identical to and accessed in an identical manner to regular user data. Thus all data stored in the system is stored in a consistent fashion making security checking, as well as access consistent for any and all data.

Finally, imbedded in the system code are facilities for monitoring program execution for debugging purposes, as well as timing of operations for system tuning. There is also an ability to log all requests made in the DML and DDL, used mainly for determining optimal data base structure. These facilities may be turned on or off in the DML.

A detailed description of the levels of implementation of the GMIS may be found in Reference 1.

Role of VM/370

The capability of running multiple virtual machines at the same time under IBM's Virtual Machine Facility/370 (VM/370)[15] has facilitated a solution to the problem of using NEEMIS as a multiple access system, with different users having varying applications requirements (e.g., report generation, econometric modeling).

In the multiple user environment, the basic requirements for a user are to send a command to NEEMIS, receive a reply that may be in a number of forms (report, single answer, return code) depending on the command, and then either displaying the reply or performing further operations on it.

These requirements are satisfied by using a single virtual machine that contains the NEEMIS data base and command processor. Each user has his own virtual machine, and communicates with the NEEMIS machine through the use of virtual card punches and shared query/reply files. User requests to the NEEMIS machine are stacked in its virtual card reader and are selected one at a time for processing. The NEEMIS machine writes the results of each request in the user's reply file, and then processes the next user in the queue on a FIFO basis.

Each user is thus provided with a reply file that can be processed by programs written in any language. Currently, programs for flexible report generation have been written in PL/I, and an econometric modeling interface that operates in an APL environment will be implemented.

Using this facility, each user can tailor his interface to NEEMIS to suit his own needs. For example, it is possible to interface TROLL, a popular econometric modeling package,[16] to NEEMIS using programs to convert NEEMIS reply files to TROLL compatible input files.

In summary, the use of multiple virtual machines facilitates increased user isolation and security,[17] multiple access to a shared data base without loss of integrity, and the capability of running many different user-dependent application interfaces simultaneously.

CONCLUSION

We have presented here a brief overview of some of the user facilities that have been made available in the NEEMIS System. These facilities have been designed with maximum flexibility and for a wide range of users in terms of both computer sophistication and type of function they perform.

ACKNOWLEDGMENTS

We acknowledge the contributions of Professor Henry D. Jacoby of the Sloan School, M.I.T., for his experience and guidance in the energy policy area.

We would also like to thank Drs. Stuart Greenberg, Paul Comba, and Ray Fessel of the IBM Cambridge Scientific Center for their insight and thoughts, specifically, Paul Comba for his guidance in preserving the mathematical and relational model of data in our DDL and DML, Ray Fessel for his ingenious programming guidance, and Stu Greenberg for his help with the VM concepts of implementation.

Since the writing of this paper, we acknowledge the contributions of members of the IBM Research Laboratory of San Jose who have greatly enhanced the operational aspect of NEEMIS, and we look forward to working with them in the future.

Work reported herein was supported in part by the New England Regional Commission (NERCOM), Boston, Massachusetts.

REFERENCES

1. Donovan, J. J. and H. D. Jacoby, *A Hierarchical Approach to Information System Design,* Report CISR-5, M.I.T. Sloan School Working Paper 762-75, January, 1975.
2. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *CACM,* Vol. 13, No. 6, June, 1970, pp. 377-387.
3. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," *Proceedings 1971 ACM/SIGFIDET* Workshop.
4. Post, E. L., "Formal Reductions of the General Combinatorial Decision Problem," *American Journal of Mathematics,* Vol. 65, 1943, pp. 197-215.
5. Church, A., "The Calculi of Lambda-Conversion," *Annals of Mathematics Studies,* No. 6, Princeton University Press, 1941.
6. Smullyan, R., *Theory of Formal Systems,* Study 47, Princeton University Press, 1961.
7. Donovan, J. and H. Ledgard, "A Formal System for the Specification of the Syntax and Translation of Computer Language," AFIPS Conference Proceedings, Vol. 31, 1967, AFIPS Press, Montvale, N.J.
8. Goldstein, I. and A. Strnad, *The MACAIMS Data Management System,* M.I.T. Project MAC TM-24, April, 1971.
9. Chamberlin, D. D. and R. F. Boyce, "SEQUEL: A Structured English Query Language," *Proceedings 1974 ACM/SIGFIDET Workshop.*
10. Bracchi, G. et al., "A Language for a Relational Data Base Management System," *Proceedings, 5th Princeton Conference on Information Science,* 1972.
11. Fehher, P. C., *The Representation of Independent Language,* IBM Technical Report RJ1121, November, 1972.
12. Dijkstra, E., "T.H.E. Multiprogramming System," *CACM,* May 1968.
13. Madnick, S. E., *Design Strategies for File Systems,* M.I.T. Project MAC TR-78, October, 1970.
14. Donovan, J. J., *Systems Programming,* McGraw-Hill, New York, 1972.
15. *IBM Virtual Machine Facility/370,* Introduction, No. GC20-1813, IBM, Burlington, Mass.
16. *Troll Reference Manual,* National Bureau of Economic Research, 1972.
17. Donovan, J. J. and S. E. Madnick, *Application and Analysis of the Virtual Machine Approach to Computer System Security and Reliability,* M.I.T. Sloan School Report CISR-2, May, 1974 (to be published in the IBM SYSTEMS JOURNAL, May, 1975).
18. Smith, G. N., *Internal Intermediate Language, Version* 2, M.I.T. Sloan School Report CISR-6, November, 1974.

Cost-benefit evaluation of interactive transaction processing systems

by GEORGE A. HOLT and HENRY C. STERN
Technology Management Incorporated
Washington, D.C.

INTRODUCTION

Interactive transaction processing systems are being designed and implemented to support an increasing number of applications in business and government. Only a small amount of attention has been paid in the literature toward providing a basis for the economic evaluation of interactive systems in general[1,2] and little of it is directly useful to a manager in a practical situation. This paper presents an approach to the economic evaluation of interactive transaction processing based on cost-benefit analysis which was developed by members of the staff of Technology Management Incorporated (TMI). This paper highlights the managerial considerations and specific analytical techniques employed in the course of evaluating a large-scale system currently under development by a major government agency. This system will automate nationwide regional office claims processing operations and will employ a distributed network of processors and remote terminals.

The approach consists of two components:

- A *framework* which is designed to structure the analysis into a set of elements capturing all significant costs and benefits pertaining to the development, procurement, installation, and operation of a system over time. The elements include not only hardware and software items but also manpower and performance effect items.
- A *methodology* which involves techniques for estimating values for the cost-benefit elements and evaluating them by means of standard investment performance tests. Each step accomplishes a specific portion of the overall analysis and by itself or in conjunction with other steps provides supporting data for other financial and technical analyses.

Although the approach was used to evaluate a highly complex system, it is equally applicable to systems of varying sizes and configurations, ranging from centralized to decentralized, and it will support aggregate or detailed levels of analysis.

MANAGEMENT CONTEXT

The development and operation of an information system should, from the manager's viewpoint, be looked upon as an investment project and subjected to the same evaluation as any other type of investment decision. The manager should understand fully the extent of the out-of-pocket and reallocated cost involved and also the nature and extent of potential dollar savings and other benefits. Too often, the decision to develop a new computer application is made without an adequate economic appraisal, either of the system itself or of alternative designs or configurations. The decision maker in this case may feel that the benefits to be gained cannot be properly evaluated in economic terms or that the task is too complex, time-consuming, and ill-structured to undertake with available staff. In general, managers have traditionally not gotten sufficiently involved in ADP matters. On the other hand, government agencies and some commercial organizations routinely require that some level of cost-benefit evaluation be made before a major ADP effort can be initiated since most systems represent a sizable investment.

Some specific considerations facing a manager interested in pursuing cost-benefit evaluation are:

Organizational responsibility—to effectively accomplish a cost-benefit evaluation, several organizational components must be represented:

- the ADP organization which has specific understanding of the system design and the relationship of hardware, software, and communications factors;
- the "user" or operating organization which understands the functional job to be accomplished, the workload, and workforce; and
- the financial organization which has responsibility for gathering financial and budgetary data across organizational boundaries with a purview of all prospective projects on which to spend resources.

The ultimate responsibility for conducting the cost-benefit evaluation should rest outside the ADP and the "user" organizations which may have vested interests and be placed with the financial organization or a separate evaluation group.

- Skills required—in addition to knowledge of the system, the functional aspects of the organization, and financial evaluation techniques, it is desirable that the group assigned to the project have abilities in computer programming, hardware and telecommunication technology, work measurement, and simulation techniques, particularly if a detailed analysis is to be done.
- Credibility of results—the ultimate credibility of the results of the evaluation will rest on the key technical and managerial assumptions underlying it. For example, the analysis should include consideration of technological risk and changes in future workload if they are expected to be significant factors. The manager must be prepared to question and test all assumptions from many viewpoints, not purely technical ones.
- Updating of results—the results of the cost-benefit evaluation should be updated over time (every six months if possible and at least yearly) to reflect internal and external changes which will affect system cost and savings factors.

ANALYTICAL CONSIDERATIONS

The economic impact of a systems project is the result of a large number of individual costs incurred and benefits received for specific activities. Normally, the bulk of the investment is made before any appreciable benefits are received, and the *time* interval between the two can make a critical difference in the value of the system. Factors which act to determine this timing are:

- length of the development phase,
- rate of system implementation (e.g., if the system has several subsystems or regional locations), and
- rate of personnel attrition (e.g., if savings are to be achieved by means of reduced personnel requirements).

The design of an interactive system must account for the *locational distribution* of the demand for processing services and the cost-benefit analysis must include the relative amounts of capital equipment, manpower, and other resources required to provide these services. These factors influence not only the cost of communication networks and terminals but also the distribution of computing resources.

Benefits which accrue from a system are included in the analysis by means of a three step process:

- identifying and defining the benefit,
- measuring the net magnitude of the benefit, and
- valuating (costing) the benefit.

Most benefits are in the form of increased productivity or reduced manpower expenditures and are said to be *tangible* since it is relatively easy to place a dollar value on them. Other benefits, such as improved service, better control, and greater flexibility which cannot be easily costed are commonly labeled *"performance effects."* They are often important in themselves and are usually evaluated by somewhat subjective means.

It is also important to define the *comparative basis* upon which the cost-benefit evaluation is being conducted. In general, a system represents either a completely new application (including extensions to existing systems) or a replacement or upgrading of an existing system. In the former case, net benefits of the project are examined to determine whether they are positive. In the latter case, net benefits must be compared to the existing system. One must also be aware of possible evolutionary changes in the existing system occurring before the new system is implemented which may change the basis of comparison.

As a final point, management should decide at the outset the *level of detail* it requires from the evaluation and the extent to which it is willing to commit resources to the task. In most cases, the focus in the early stages of a project is on *aggregate* questions, i.e., total investment, total cost savings, return on investment, and organizational impact. As the project progresses, system design becomes more detailed and so must management planning. Consideration must be made of hardware selection, site preparation, changes in personnel policy, etc. To answer this kind of question, the evaluation must become more *detailed.*

THE ANALYTICAL FRAMEWORK

The analytical framework developed for cost-benefit evaluation of interactive systems is designed to encompass all significant costs and benefits related to a system during its development and operational lifetime. Its primary use is in defining the boundaries within which investigation of various costs and benefits is undertaken.

Costs are classified into three major categories:

1. One-time development costs—include manpower costs associated with systems development, procurement, installation, and training; travel costs; subcontracting costs; and site preparation costs.
2. Hardware/software costs—include the purchase of computer hardware (CPU's, terminals, and peripherals) and proprietary software packages.
3. Recurring operational costs—include the manpower costs associated with the operation and maintenance of the system; outside maintenance contract cost; utilities; and the annual cost of any system hardware which is leased, such as communication lines.

One-time development costs and hardware costs are considered to be the *investment cost* of the system.

Benefits are also classified into three major categories:

1. One-time cost recovery—includes the salvage value of eliminated or replaced facilities and equipment.

2. Recurring tangible savings—include savings (both positive and negative) in costs resulting from operation of the system due to modifications of processes and procedures as well as elimination of manual procedures through automation.
3. Performance effects—include all recurring benefits which are not readily convertable into dollar amounts, such as improvements in quality of service, timeliness, control, responsiveness, and flexibility.

The difference between recurring savings and recurring costs is the *net annual savings* (positive or negative) resulting from the system during its operational lifetime.

Table I presents the structure of the analytical framework and lists the major cost-benefit elements which fit into each category.

THE EVALUATION METHODOLOGY

Once the system cost-benefit elements have been outlined, the remainder of the evaluation effort lies in developing values for them and then applying an investment performance test to those values. There are two basic levels of analysis which can be undertaken in the evaluation process:

- "top-down" (aggregate), and
- "bottom-up" (detailed).

In the top-down approach, the strategy is to examine aggregate cost elements of the existing system in order to identify similarities and differences between it and the proposed system. The present and proposed systems are considered to have the same demand pattern or workload which simplifies the task by eliminating the problem of determining the impact of workload changes on system size and operational costs. The major cost elements of the system are classified into five categories according to functional impact:

- functions partially or totally eliminated,
- revised procedures within a function,
- new functions,
- functions with little or no change, and
- overhead and management functions.

In each case, an attempt is made to estimate the impact of the proposed system on the cost of the function. Costs which are substantially unchanged are carried over directly. Elements that differ are examined more closely to determine the magnitude of change. This approach stresses simplicity and opportunism in both data collection and analysis. The key objective is to develop lower bound savings and upper bound cost estimates in order to establish the economic feasibility of the project.

The bottom-up approach, in contrast, stresses greater accuracy and completeness and is more appropriate for the later stages of cost-benefit evaluation. In this approach, the sequence of methodology steps which are followed are outlined below and explained in more detail in the next section.

TABLE I—Analytical Framework

COSTS	BENEFITS
One-Time Development System Development • software design and programming • customizing purchased software • software testing/integration • software documentation • operating & user procedure design & documentation • outside contractors • purchased computer time • travel System Procurement • request for proposal (RFP) development • RFP evaluation and vendor selection • vendor negotiation and contracting System Installation • conversion • site construction & preparation • acceptance testing Training • operating personnel • user personnel Hardware/Software Hardware Purchase • CPUs • terminals • peripherals Software Purchase • data base packages • telecommunication packages Recurring Operational System Operation • operators and supervisory personnel • facility rental and utilities System Maintenance • vendor or in-house hardware maintenance • system program modification and maintenance Hardware Leasing/Rental • CPUs • terminals • peripherals • communication lines	One-Time Cost Recovery Salvagable Equipmt. & Facilities • eliminated equipment and facilities • replaced equipment and facilities Recurring Tangible Savings Eliminated Functions • completely automated manual operations • reduced demand on manual operations • outside contracted services Revised Procedures • improved efficiency & productivity effects Performance Effects Quality of Service & Goodwill Throughput Timeliness Operational Control Demand Responsiveness Operational Flexibility Personnel Morale

1. Size the workload—by type, long-run volume trends, geographical distribution, and seasonal factors.

2. Define the workflow—in terms of the processing to be utilized in conjunction with the system, i.e., the work steps performed at each stage of processing.
3. Define and size system interactions—in terms of the volume of man-machine interactions required to support the work steps. Interaction volumes are measured in terms of commands, message transmissions, data base accesses, etc.
4. Size the processor capacity—determine the size and number of processor units required with respect to effective computation rate, storage capacity, I/O channel capacity, operating system overhead, and response time.
5. Size the network capacity—with respect to geographical distribution and the number of terminals and communication lines.
6. Determine the resulting manpower changes—in the various categories of personnel affected by implementation of the system, including System Operation and System Maintenance personnel.
7. Valuate the resources required—for manpower and data processing including the costs of System Development, System Operation, Maintenance and Installation, System Procurement, Hardware and Software Purchase/Leasing, and Training; Cost Recovery; and Tangible Savings.
8. Analyze investment performance—based on the cost-benefit data developed in the previous steps and measured in terms of payback period, return on investment, or discounted net benefits.

The flow of the evaluation steps is illustrated in Figure 1.

DETAILED EVALUATION TECHNIQUES

The purpose of this section is to examine the eight steps of the evaluation methodology in terms of the bottom-up approach and to briefly describe some of the specific techniques which can be employed in performing them. In examining the collection of cost-benefit elements outlined in the framework, it is evident that certain ones are more difficult to valuate than others. Among them are hardware and network purchase and leasing costs and recurring tangible savings. The majority of the techniques described in this section are oriented toward valuating these elements.

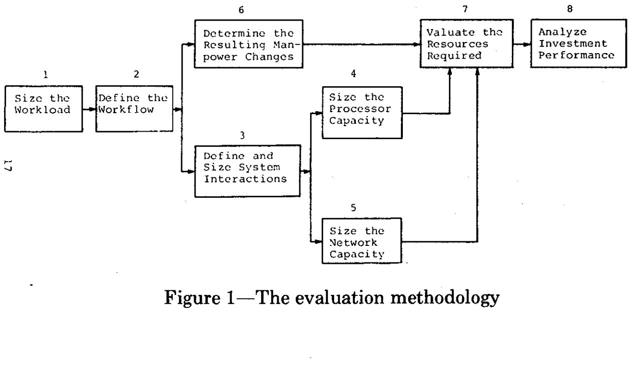

1
Size the Workload
2
Define the Workflow
6
Determine the Resulting Manpower Changes
3
Define and Size System Interactions
4
Size the Processor Capacity
5
Size the Network Capacity
7
Valuate the Resources Required
8
Analyze Investment Performance

Figure 1—The evaluation methodology

Sizing the workload

Workload sizing is based primarily on analyzing historical volumes for each work item which will be processed on the system. Several years of historical data can be examined to develop daily, weekly, or monthly seasonal indices and geographical distribution factors. Long-run trend projections of workload over the expected system lifetime can be made by using standard forecasting techniques such as regression analysis, exponential smoothing, Box-Jenkins, and spectral analysis, or structural models which account for various external factors.

The main objective of this step is to determine both long-run growth patterns in the workload and the peak workload. This step provides a basis for assuring sufficient overall system capacity and for making trade-off decisions concerning the value of being able to handle peaks efficiently.

Defining the workflow

The purpose of this task is to define how the workload will be processed in terms of the work steps involved. Each type of work item requires work to be done at one or more processing stations and the type of procedures involved may vary from item to item. More than likely, introduction of the system will cause changes to be made in many of the established procedures, replacing manual operations with man-machine interactions. What is required in this task is to:

- define the flow of work through the work stations,
- define the new procedures for each item and for each station,
- identify who will perform the various procedures, and
- estimate the average amount of time involved for each step.

If the workflow is straightforward, it is sufficient to list the stations and their associated personnel types and processing times by work item. If the flow involves a number of stations with many possible inter-routings and personnel types, it may be desirable to construct a dynamic simulation model which represents the stations and flows. A model of this nature has several advantages:

- it is easily changeable and usable,
- it allows observation of behavior under varying workload volumes and personnel assignments, and
- it can be designed to generate statistics which might otherwise be unavailable, such as queue lengths and work completion times.

Several programming languages are suited for this type of application, such as GPSS and SIMSCRIPT II.

In either case, this step provides a means of translating the workload projections into the volume of worksteps and the number of manhours involved in processing. This data provides the input for the manpower changes and system interaction steps.

Defining and sizing system interactions

Each work step identified in the preceding step will involve a combination of manual subtasks and man-machine interactions. The purpose of this step is to examine the man-machine interactions in order to assess the work that the system itself must perform in order to support the work steps. The system work is characterized as a sequence of "system work elements." These elements are defined at an intermediate level between those of instructions and applications modules. Each interaction sequence begins with either an operator-initiated command or a prompting message from the system. The composition of the set of work elements depends upon the particular system, but almost any set would contain the following major categories:

- operator commands (terminal to processor),
- screens or responses (processor to terminal),
- messages (processor to processor),
- file access and maintenance (processor to storage),
- on-line printing (processor to peripheral), and
- miscellaneous routines (non-trivial processing).

The easiest way of constructing the interaction sequences is to use high or intermediate level flowcharts. In cases where there are optional commands, screens, etc., which can be used, estimates must be made of their frequency of occurrence. The output of this step is a set of all possible interaction sequences for each work step with a frequency of occurrence value for each element in the sequence.

Sizing processor capacity

The amount of processor capacity needed to handle a given amount of interactive work is a function of:

- the volume of work elements of various types which must be performed within a specified time interval, and
- the amount of capacity utilized in performing a given work element.

The workflow and system interaction analysis provide the data for the first requirement. The amount of processor capacity required per work element is based on the number of machine instructions executed per element. For example, each of the elements outlined in the previous section would require in the range of 1000 to 2000 instructions in a typical data base and telecommunication processing environment. The total number of instructions to be processed is found by summing across all work elements the products of the element frequency and its instruction set length.

Several adjustments can be made to these results:

- In order to insure fast response time, CPU utilization should be kept low (for example, in the range of 25 to 30 percent).
- If processing capacity is to be distributed among several sites, the larger relative workload fluctuation per site requires greater capacity per site than just the proportional fraction of total capacity.
- Any provisions for failsoft and backup capability will require additional capacity.

System storage requirements are of two types—processor (core) and data base (disk, extended core, etc.). Processor storage needs in many systems fall into three categories:

- static—used by the operating system and user-shared re-entrant and serially reusable application programs,
- user dynamically allocated—allocated on a per user basis to hold user session data, and
- terminal dynamically allocated—allocated to hold intermediate data during transmission to and from a terminal.

The need for static storage is dependent upon the particular hardware, operating system, data base package, and telecommunication monitor employed. About one million bits are typically needed on average for each operating system or package in operation. The amount of user dynamically allocated storage depends upon the average number of users at a given time while the amount of terminal dynamically allocated storage is dependent upon the average number of commands and screens transmitted to and from the terminals at a given time. Data for these requirements can either be roughly estimated or be obtained by using a contention model similar to the one used in the network capacity analysis.

The amount of data base storage required depends to a large extent upon the type of data involved. If the data base is stable in nature, such as for a master file, the size is relatively easy to estimate. If the data base is dynamic in nature, such as for a work-in-process file, the size is dependent upon the volume of new records created per unit time, their average size, and their average length of retention.

Other peripheral capacity requirements, such as for printers, can be estimated in a similar manner. More detailed hardware sizing can be accomplished by using the data developed in this step in conjunction with a hardware simulation system such as SCERT or CASE.

Sizing network capacity

The communication channel capacity required for a system depends on the volume and size of messages to be transmitted and on the amount of delay in transmission that is acceptable. The volume of messages is an output of the system interaction analysis. Transmission delays result from the unavailability of a line if it is shared by several sources and by the capacity of the line itself.

An aggregate level estimate can be made by estimating the traffic between each point in the network in terms of bits, characters, messages, or CRT screens transmitted per unit time and translating it into the number and size of the lines required between each point. Total network costs are found by calculating mileage and non-mileage (modems, line conditioning, connect charges) costs based on the number and size of the lines.

A more refined technique is to construct a communication link contention model of the network which incorporates:

- the mean message arrival rate at each sending point in the network,
- the mean length per message, and
- the desired wait probability and duration.

By varying the performance constraint while holding the other conditions constant, the model provides a means for evaluating the capacity required to achieve alternative levels of performance. Details of the model cannot be explained here, but it is based upon a queuing theory approach.[3,4]

Terminal requirements for a system depend upon the volume of inputs and screens transmitted and the average amount of operator time taken per input or screen. The volume depends upon the type of work items being processed and this information is an output of the system interaction analysis. Terminal usage time is composed of four basic types of operator actions:

- reading time to interpret data displayed on a CRT screen or printed page,
- retrieving data from source documents to be input to the system,
- entering data on the keyboard, including decision time, and
- session preparation, logon/logoff, and other miscellaneous activities.

System response delays also may be included. In any event, it is necessary to either estimate or conduct a detailed analysis in order to arrive at average session times per work step. A contention model, similar to the one used in the communication link analysis, can be used with this data to determine the number of terminals required, given the amount of terminal sharing and waiting time that is acceptable.

Determining manpower changes

Recurring tangible savings are in most instances the result of a change in the required manpower skill mix caused by introduction of the new system. (If the application is new and the personnel are purely additional, then the benefits of the system lie in the realm of performance effects and earned revenues, if any, since manpower will be a cost item.) The workflow analysis and/or model serve to project manpower under the new system. Overall manpower changes, then, are simply the differences between projected manpower in the new system and projected manpower in the current system in each affected job category. The only difficulty that might be encountered here is that current unit manpower requirements may not be known with any accuracy. In this case, it would be necessary to establish work rate standards for each task that is affected.

In this analysis, several points are worth mentioning:

- In practical situations, personnel levels cannot always be cut immediately. Instead, attrition due to retirements, resignations, and transfers must act over a period of time. Allowance for this factor should be included in order to prevent overstating savings.
- The system may not be implemented at all remote locations simultaneously. If the system is phased in gradually, savings should be similarly adjusted.
- While introduction of the system may eliminate work that would otherwise continue to be done in the existing system, it may also result in additional work not presently required. Care must be taken to estimate both types of effects.

Manpower needed for system operation and maintenance is dependent upon the size and number of processor sites.

Valuating resources

This step involves determining unit costs for hardware, software, and manpower and applying them to the various capacities determined in the previous steps. For the hardware, it must be decided what equipment will be purchased and what will be leased. Significant economies of scale exist for processors in terms of cost-effectiveness in instruction execution. This economy is sacrificed to some extent as processing power is distributed across several sites. The same effect holds for other hardware items.[5] For manpower, the increase in *real* wages during future time periods may be included. Historically, real wages have increased at two to three percent annually.

Costs for other items such as system installation, procurement, and maintenance are functions of the particular system configuration selected. Training and software purchase costs are also unique to the system and its design.

Analyzing investment performance

The final step in the process is to analyze the cost-benefit data in terms of investment performance. Several

TABLE II—Discounted Net Benefits Analysis

			($000s)			
Benefits	1975	1976	1977	1978	1979	1980
Eliminated Functions						
Keypunching	0	0	311	2370	2370	2370
Teletype Messages	0	0	118	869	869	869
File Look-ups	0	0	680	4959	4959	4959
Revised Functions						
Records Processing	0	0	1299	5491	5491	5491
Accounting	0	0	87	607	607	607
Gross Benefits	0	0	2490	13996	13996	13996
Costs						
System Hardware	0	4350	0	0	0	0
CRT Terminal Hardware	0	0	627	2508	2508	2508
Communication Facilities	0	0	214	858	858	858
Computer System Operations	0	0	232	464	464	464
System Development & Installation	2420	2422	956	0	0	0
Total Costs	2420	6772	2029	3830	3830	3830
Net Benefits	(2420)	(6772)	461	10166	10166	10166
Discount Factor @ 10%	1.000	0.909	0.826	0.750	0.681	0.619
Discounted Net Benefits @10%	(2420)	(6156)	381	7625	6923	6293
TOTAL DISCOUNTED NET BENEFITS @ 10%	12646					
BENEFIT-COST RATIO	1.70					

(NOTE: This analysis is intended as a simplified example which is carried out for only a portion of a typical system lifetime and assumes constant workload and manpower costs.)

techniques exist for this purpose, among them are:

- discounted net benefits (net present value),
- payback or breakeven analysis, and
- return on investment.

An example of a discounted net benefits analysis which represents the end product of an evaluation is illustrated in Table II. A sophisticated approach would include the aspect of risk or uncertainty in the estimated cost-benefit element values. This can be done by making optimistic and pessimistic evaluations or by assigning probability distributions to the values. The second investment measure, the payback period, is computed as the ratio of the investment cost to the net annual savings if the net annual savings is relatively constant over time. Otherwise, its computation is slightly more complex. The return on investment appears in Table II as the Total Discounted Net Benefits figure calculated at ten percent in this example.

If alternative system designs or configurations are being evaluated, a separate analysis should be done on each one and each ranked by its respective benefit-to-cost ratio. At this point, management has the necessary data to make a go/no-go decision or to select the appropriate option, taking into consideration the related performance efforts.

EXTENSIONS TO RELATED SYSTEM DEVELOPMENT AREAS

The data developed in the course of the cost-benefit evaluation is useful in other areas of the system development effort. These include:

- Manpower and operations planning—the manpower changes data can be used to forecast future personnel level requirements and to develop the personnel policies necessary to achieve them. The workflow model can also be used as a basis to optimize work procedures.
- Budgeting—the cost and savings data when arranged on a fiscal year basis can directly support budget forecasting requirements.
- Procurement—hardware and software performance data developed in the course of the analysis can be used to support Request for Proposal preparation and benchmarking requirements.

ACKNOWLEDGMENTS

The authors would like to acknowledge the contributions of Joseph P. Mazzetti, G. Allen Moulton, and C. Wade Tambor to the project work upon which this paper is based.

REFERENCES

1. Sharpe, W. F., *The Economics of Computers,* Columbia University Press, New York, 1969.
2. Streeter, D. N., "Cost-Benefit Evaluation of Scientific Computing Services," *IBM Systems Journal,* Vol. 11, No. 3, pp. 219-233, 1972.
3. Morse, P. M., *Queues, Inventories and Maintenance,* Wiley, New York, 1958.
4. Martin, J., *Systems Analysis for Data Transmission,* Prentice-Hall, Englewood Cliffs, New Jersey, 1972.
5. Streeter, D. N., "Centralization or Dispersion of Computing Facilities," *IBM Systems Journal,* Vol. 12, No. 3, pp. 283-301, 1973.

A note on recoverability of modular systems*

by PHILIP M. MERLIN and DAVID J. FARBER
University of California
Irvine, California

INTRODUCTION

In a paper by Gostelow-Van Weert,[1] it was shown that any processes can be described by Petri-nets (PN). Thus PN can be used as a model in the design (or analysis) of any computer system. This approach makes it possible to utilize all the theoretic knowledge developed for PN and thus provides a powerful tool for computer systems (or program) design. In this case, the PN can be used as a framework that provides a tool for:

1. the understanding of the design,
2. handling the incremental changes of the design (without redesigning the entire system),
3. the implementation of desired properties (like recoverability in case of failures, proper termination, mutual exclusion, etc.), and
4. at least, the analysis of the program in order to check if it has the desired properties.

In Reference 2, it was shown that the UCLA Graph Model of Computation and the Petri-nets are equivalent. This means that the same approach can be applied using PN, the UCLA Graph Model, or any other equivalent model of computation. Note that these models are able to represent a concurrent environment (multiprocessing, multiprograming, computer networks, etc.).

In the past, the use of these models as a practical tool was limited. The principal reason of this limitation is that any program (or system) of reasonable size has a corresponding very large model, usually of unacceptable size, even for processing by computer.

In order to simplify the handling of these models of programs, many researchers tried to find a way of dividing the models into modules. We note that the models of structured (modular) programs are easily divisible into natural modules, the subroutines. Then, any *structured, hierarchical, top-bottom, modular design* can be relatively easily modeled by Petri-nets, et al. At each level, the description of the desired system behavior is given. This description is then modeled by a Petri-net and analyzed using the theory developed for Petri-nets.

We note that distributed computation naturally leans toward structured (modular) programming and indeed makes it difficult to do otherwise. We feel that the application of the above modeling methodology will enable one to gain strong insight into this and other areas.

In the following sections, we will focus our discussion on the problem of recoverability. Our approach is based on the method for analyzing and designing recoverable Petri-nets presented in References 3, 4 and 5. The same method is improved in Reference 6, and an example of its use is given in Reference 7. By "recoverability" we mean that after the occurrence of a failure the control of the process is not lost, and after several steps it will return to "normal execution". Note the difference between this concept and the concept of "correctness of results". In this paper we do not deal with the problem of correctness. We are concerned with control recoverability—structural recoverability. Our approach is motivated by the philosophy that we can accept the situation in which a user gets some erroneous results, but we do not accept the possibility that a single error (or failure) may cause the entire breakdown of big systems.

In the next section a short background is presented. A later section presents an example, the top level design of a simple operating system.

RECOVERABILITY OF A PETRI-NET

Petri nets[8] model "conditions" represented by *nodes* and "events" represented by *transition bars.* The holding of a condition is represented by placing a token on that node. *Directed arcs* connect nodes to bars and bars to nodes. A transition bar (event) can fire (occur) if all the nodes (conditions) input to that transition bar have tokens (hold). When a transition bar fires, it removes one token from each input node and places one token on each output node. If we permit repetitions of the directed arcs, the number of tokens placed (or removed) in each node is equal to the number of repetitions of the arc connecting the transition bar and the node.

Figure 1 shows a PN example. Assuming as initial condition a token in node 1, bar 1 can fire. When bar 1 fires it removes the token of 1 and places a token on 2 and two tokens on 3. In this new state (2, 3, 3) only bar 2 can fire. If bar 2 fires it removes a token from 3 and places a token on 4. In this position bar 2 or bar 4 can fire, and so on.

The state of the Petri-net is defined by the collection of names of the nodes holding tokens. The number of

* This work was supported in part by the National Science Foundation under grant GJ-1045—"The Distributed Computer Project".

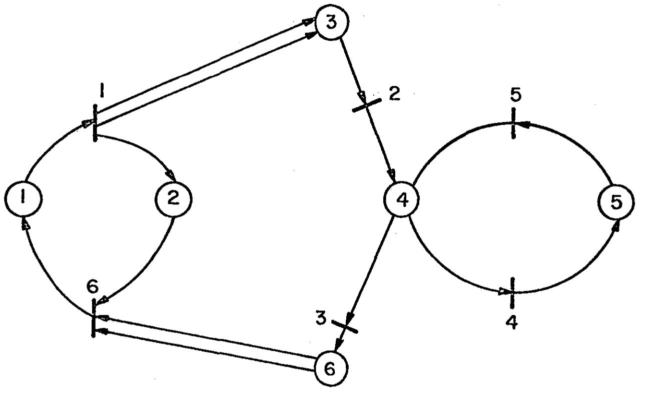


Figure 1—A Petri-net

instances of a node in a state is equal to the number of tokens the node holds in this state. All the possible states in which a PN can stay and the possible transitions between them define a state machine called *Token Machine* (TM). The TM for the net of Figure 1 is shown in Figure 2, assuming as initial state a token in 1.

Suppose that a condition in the PN may fail. In this case, a token held by this condition may disappear. It is possible to represent this characteristic by adding a new branch to the TM. This branch will represent the possible flow of the execution when the "problematic" token disappears. For illustration, suppose that when the PN of Figure 1 is in state (2, 4, 5) a token of 5 disappears, then the PN will go to state (2, 4). In state (2, 4) bars 3 and 4 can fire. A fine state machine including the TM of figure 2 and all the additional paths for the case that 5 can lose a token is given in Figure 3. Thick lines represent the TM and thin lines describe the paths added since a failure. This new machine will be called *Error Token Machine* (ETM).

The states (and transitions) of the TM are called "legal states" ("legal transitions") and the other states (transitions) are called "illegal states" ("illegal transitions").

At this point we can state the conditions for recovery:

> "A process P is recoverable from failure F if and only if in the ETM of P for failure F, *all* the directed paths through illegal states arrive to legal states."

It means, after a failure, the execution sequence must return to normal execution after a finite number of steps.

In this work, we limit our study to processes that have finite TM. From the properties of directed graphs, for the case of finite TM we can derive an equivalent set of conditions for recoverability:

> "a process P is recoverable from failure F if and only if in the ETM of P for failure F:
> 1. the number of illegal states is finite,
> 2. there are no final illegal states, and
> 3. there are no directed loops including *only* illegal states."

In this paper, these conditions are named *Conditions of Recoverability* (COR). Figure 3 shows that the PN of Figure 1 is not recoverable from failures in 5. Conditions COR are not satisfied because of the final illegal state (2, 6) and because of the loop between the states (2, 4) and (2, 5).

In the previous discussion we deal only with the case that a token may disappear. But in the same way, because of a failure, a condition in the PN may *generate* a token. This situation may also be represented by adding new branches to the correspondent nodes in the TM. The approach is similar to the previous case.

In References 4 and 6, it was shown a method of designing recoverable PNs for failure of kind "loss of token". The design is executed in two steps. First, a TM that can be implemented by a recoverable PN is designed. In the

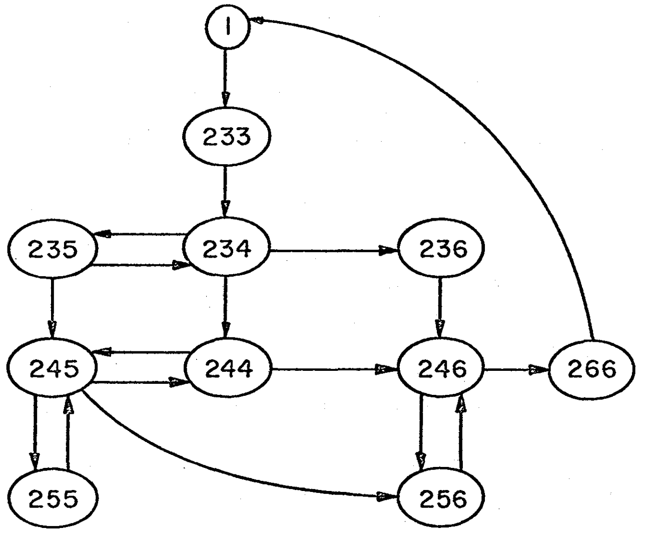


Figure 2—The TM for the PN of Figure 1

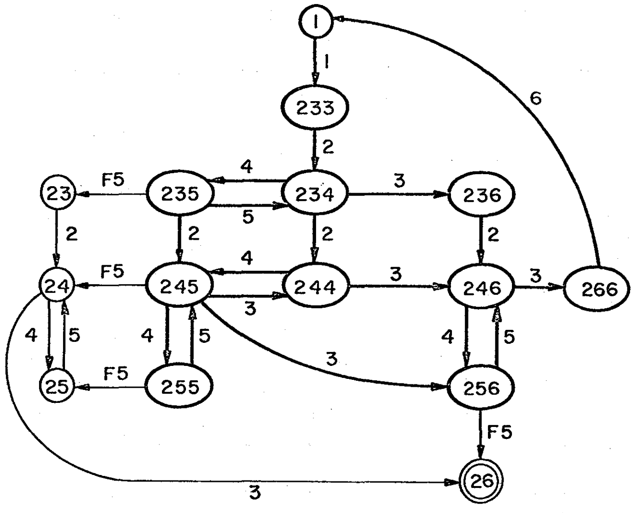


Figure 3—The ETM for the PN of Figure 1

second step, a recoverable PN corresponding to the given TM is generated. In order to accomplish this method,[4,6] develop a way of designing PNs that affect a given TM, and also gives the necessary and sufficient conditions that a TM has to satisfy in order to be implementable by a recoverable PN. Since the TM of Figure 2 contradicts these conditions, this TM cannot be implemented by a recoverable PN.

In References 5 and 6, it was shown that in case where no assumptions have been made about the execution times of the different parts of the PN, the recoverable processes under a failure of type "loss of token" are very limited in their possible structures. These limitations are usually unacceptable in practical processes. Because of these limitations, some knowledge about the execution times was introduced in the PNs, and a new model, the TPN, was defined. In this new model, practical processes can be implemented as recoverable, but in this recoverable TPNs it is necessary to state some constraints on the execution times of its parts.

The reader interested in a more detailed and formal discussion on the TPN and recoverability of the TPN is referred to the sources.[5,6] Here we limit ourself only to the discussion "how to design a recoverable TPN that implements the TM of Figure 2".

A TPN is defined by a PN where for each transition bar b_i a pair of real numbers (t^*i; $t^{**}i$) are given. In a TPN a bar b_i can fire only after its input conditions hold for a period of time larger than t^*i. On the other hand, if the input conditions of bar b_i hold for a period of time $t^{**}i$, then b_i *must* fire. Note that in some sense, t^*i and $t^{**}i$ give a measure of minimal and maximal execution times of the events (the executions). Note that when no constraints are stated for these times (for all i $t^*i=0$ and $t^{**}i=$infinite) the TPN is equal to a PN.

In order to transform the PN of Figure 1 into a TPN that satisfies the conditions COR and has the TM of Figure 2 two changes have to be introduced. First, the illegal loop of states (2, 4) and (2, 5) have to be eliminated. Second, (2, 6) has to be converted into a no final state.

The loop of states (2, 4) and (2, 5) is broken if bar 4 is split into four different bars, as shown in Figure 4. Note that the TM of the PN of Figure 4 is the same TM of Figure 2, but its ETM (see Figure 5) differs from the ETM of Figure 3. References 4 and 6 present an analysis on the ways of breaking illegal loops.

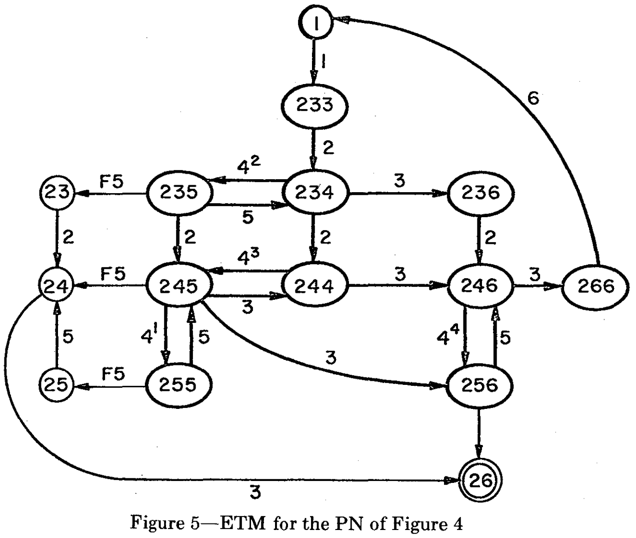


Figure 5—ETM for the PN of Figure 4

In order to transform state (2, 6) into a non- final state it has to be a bar, say 7, that fires in state (2, 6) bringing the system to one of its legal states. For example, bar 7 can be defined as:

input conditions of 7 = 2, 6
output conditions of 7 = 1

This new TPN is shown in Figure 6. But, if the TPN of Figure 6 is supposed to have the same TM of Figure 2, then bar 7 has to be prevented from firing in any state out of (2, 6). In other words, t^*7 has to be bigger than the maximal time that the conditions 2, 6 can be enabled in the TPN. If this time is denoted as $T^{**}26$, then t^*7 has to satisfy:

$$t^*7 > T^{**}26$$

In this case, bar 7 can fire only after it is sure that the TPN is in state (2, 6).

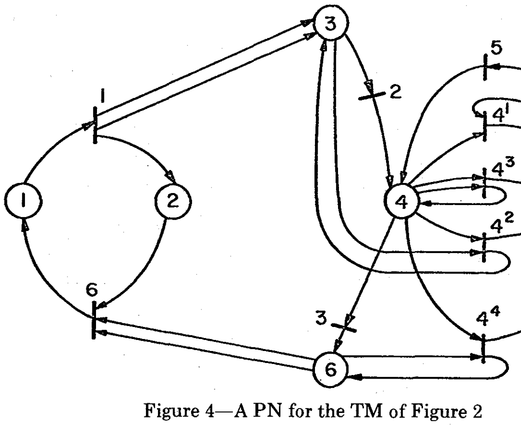


Figure 4—A PN for the TM of Figure 2

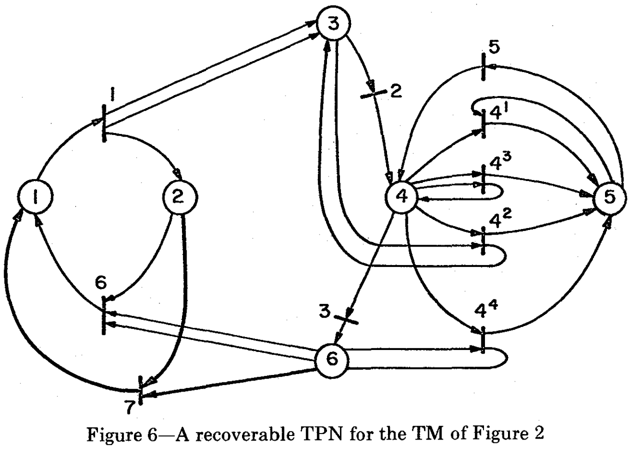


Figure 6—A recoverable TPN for the TM of Figure 2

It is possible to show that $T^{**}26$ satisfies:

$$T^{**}26 \leq \max(t^{**}2+\min(t^{**}3;\ t^{**}4^{4}) + t^{**}6;\ t^{**}5+\min(t^{**}3;\ t^{**}4^{4})+t^{**}6)$$

Thus, if:

$$t^{*}7 > \max(t^{**}2+\min(t^{**}3;\ t^{**}4^{4})+t^{**}6;\ t^{**}5+\min(t^{**}3+t^{**}4^{4})+t^{**}6) \quad (1)$$

then:

$$t^{*}7 > T^{**}26$$

is satisfied.

A general study on the problem of eliminating final illegal states is presented in References 4, 5, and 6.

The TPN of Figure 6 with the constraint (1) has the TM of Figure 2 and it is recoverable after the occurrence of a failure of kind "loss of token" in condition 5.

The next section demonstrates a practical example of the use of the methodology presented.

EXAMPLE—THE DESIGN OF A SIMPLIFIED O.S.

In order to present an example big enough to show our methodology, but small enough to fit into the physical limitation on papers for this conference, we decided to present a simplified version of an operating system (OS) design. We admit that our example is much simpler than any "real" OS, but it shows the way of designing we advocate. The designer of any "real" system can go the same path in the design of a recoverable system.

We advocate modular top-down design. The system defined at each level is modeled by a Petri-net. In this paper, we only present the top level design of the OS. We assume that the same approach is applied to the other levels.

Suppose that our simplified operating system is described as following:

1. The system loads two jobs, executes them, and only when both are finished it can load two other jobs.
2. The system can be in one of two main states: WAIT or BUSY.
3. In WAIT two jobs can be loaded, and then the system goes to the state BUSY. (Note that in this configuration, the operating system can also execute one job at a time by loading a job and a "null" job.)
4. In the state BUSY, the jobs are executed and when both are finished, the system goes to the state WAIT.
5. After the jobs are loaded, each one is executed in the following steps:
   (1) the operating system starts the job (by setting initial values, allocating resources, etc.),
   (2) the operating system checks if the job is finished. If the job is not finished, the OS executes monitor subroutines as necessary (tests, resource allocation and deallocation, etc.,) and goes to the actual execution of the user program (5.3).
   (3) The user program is executed until a "call monitor" is found. Then step 5.2 (above) is executed.

   These execution steps can be time sliced for the two jobs.
6. When the two jobs are completed, the OS goes to the state WAIT.

We note that the previous OS description corresponds to the PN of Figure 1 (or the TM of Figure 2) when the following interpretations are given to the conditions and bars:

condition 1=WAIT
condition 2=BUSY
condition 3=JOBS READY
condition 4=OS MONITORING
condition 5=JOBS IN EXECUTION
condition 6=JOB COMPLETED
bar 1=JOB LOADER
bar 2=INITIALIZE JOB
bar 3=TERMINATION CHECK
bar 4=MONITOR SUBROUTINE
bar 5=USER PROGRAMS
bar 6=CHECK FOR TERMINATION OF TWO JOBS

Each bar denotes a subroutine (or a part of a subroutine), the distribution of tokens among the conditions denotes the state of the system, and the arcs represent the control structure.

Suppose that the user program executes input (or output) and then *wait* for interrupt. An example of a temporary failure would be that the input device never sends an interrupt request, or that the interrupt request was sent, but because of a failure it was not actually executed. In all these kinds of failures (and others) the user program will remain waiting forever. Thus, bar 5 (Figure 1) will be using the "resources" provided by condition 5, but it will never fire. This situation can be represented as a "loss of token" in condition 5.

The recoverability after "loss of token" in condition 5 was presented in the previous section. As explained there, the process represented in Figure 1 *cannot be recoverable if there is not introduced some limitations on the execution times of the associated events.* The recoverable TPN was given in Figure 6. Bar 4 was split into four different bars that can fire only if a failure has not occurred. In other words, before the OS allocates a token to condition 5, it checks the system status in order to find out if any token was lost. In case of failure, bar 4 cannot fire and thus bar 3 will fire. After a time long enough (see inequality (1)) the "recovery routine" (bar 7) is activated returning the system to normal initial state. Note that in this case of failure, one or two jobs can be destroyed, but the system recovers and jobs can be executed again.

Note that the system is recoverable only if inequality (1) is satisfied. Thus, the maximal execution time between successive "monitor calls" from the user programs (t**5) *must* be finite. The other times appearing in (1) are usually known and are finite.

SUMMARY

We have demonstrated a methodology for designing and checking a system for certain recoverability properties. In order to do this, it was necessary to accept constraints on the execution time of its parts.

We believe that the method presented can be used as a practical tool. In order to apply the method efficiently, the designer is urged to refer himself to the references of this paper for a wider mathematical background of the approach used.

REFERENCES

1. Gostelow, K. P. and T. J. Van Weert, "Processes and Networks," Stichting Academisch Rekencentrum Amsterdam, Postbus 7161, Amsterdam, the Netherlands, January 1974. (Author's present address: Department of Information and Computer Science, University of California, Irvine California 92664).
2. Gostelow, K. P., *Flow of Control, Resource Allocation, and the Proper Termination of Programs,* Ph.D. Dissertation, ENG-7179, Computer Science Department, University of California, Los Angeles, December 1971 (UCLA-10P14-106).
3. Merlin, Ph. M., *Recoverability of Processes,* Technical Report #44, Department of Information and Computer Science, University of California, Irvine, California 92664, February 1974.
4. Merlin, Ph. M., *A Study on Recoverability of Processes,* Technical Report #47, Department of Information and Computer Science, University of California, Irvine, California 92664, April 1974.
5. Merlin, Ph. M., *The Time-Petri-Net and the Recoverability of Processes,* Technical Report #48, Department of Information and Computer Science, University of California, Irvine, California 92664, May 1974.
6. Merlin, Ph. M., *A Study on the Recoverability of Computing Systems,* Ph.D. Dissertation, University of California, Irvine, in preparation for publication.
7. Merlin, Ph. M. and D. J. Farber, *A Note on Recoverability of Modular Systems,* Department of Information and Computer Science, University of California, Irvine, California 92664, June 1974.
8. Holt A. W., H. Saint, R. M. Shapiro and S. Warshall, *Final Report for the Information System Theory Project,* Rome Air Development Center (Applied Data Research Inc.) Contract #AF30(602)-4211, 450 Seventh Ave., New York, New York 10001, 1968.

An integrated approach to network protocols

by LOUIS POUZIN

Institut de Recherche d'Informatique et d'Automatique
Rocquencourt, France

INTRODUCTION

Host-to-host protocols (H-H) for heterogeneous computer networks are still in infancy. So far very few implementations are in existence. Among those on which documentation is available are Arpanet[1] and Cyclades.[2] The former provides only for basic services allowing the transfer of up to 1000 octet messages, with flow control but not error control. The latter allows up to 32 000 octet messages, with error and flow control. Both are similar in the sense that they offer only a message transfer service, which is intended for building higher level protocols more appropriate for specific uses. Since data to be transferred are usually structured in various ways, a traditional approach is to superimpose additional layers of specific protocols, each one dealing with a particular level of structure. While being functionally correct, this approach leads to heterogeneity, redundancy and overhead among the various layers.

Part of the difficulty can be attributed to the heterogeneity of operating systems, for which there is no well accepted common model. Another reason is the mistiness of the host concept, which is usually assumed to be of a certain kind by protocol designers. There is no doubt that a big enough computer can support a protocol of several thousands of instructions. But what if a host is simply a terminal? It should be kept in mind that communication between an application program and a terminal must wind its way through an H-H protocol somewhere along the path. If this protocol logic is rather involved, it will be economically necessary to share it between a number of terminals. Hence the need for concentrators, which may be considered as mini-hosts, (Figure 1). But is this the only desirable solution?

TERMINAL ACCESS

Most existing networks are actually star networks of terminals. In the coming years they will tend to adopt packet switching, and at the same time introduce more efficient and homogeneous end-to-end protocols (E-E) no longer tied up with the intricacies of the communication gear. Application programs and terminals will become independent of communication systems.

By integrating several overlapping star networks there will be a need for computer-to-computer communications. But the most frequently required type of communications will be terminal access. This trend is consistently pointed out by various forecasts, such as Eurodata. Consequently, the development of networks will be strongly influenced by the overall cost of accessing terminals, including all communications gear interposed between computers and terminals.

Minimizing the cost of terminal access includes reducing line mileage and bandwidth, as well as intermediate intelligence necessary to relay communications. The minimum logic required by a terminal is a transmission procedure, if the traffic is to be sufficiently reliable. Line mileage and bandwidth are reduced with multi-point local networks using packet mode traffic. Loop[3] or tree networks[4] make terminals accessible through a ramification of inexpensive links relayed by packet concentrators and base-band modems. This should be more economical and flexible than traditional multiplexing or concentration, (Figure 2).

At this point, one may wonder what is a host. Any terminal could be a host, but this is not very economical in terms of address space, since the address field would have to be tailored for a much larger population than is usually anticipated for hosts. Another approach would be to interpose a mini-computer as a front-end to the communication network for the sake of playing host to the terminal cluster, (Figure 3). This bogus host could indeed perform its part of an H-H protocol, but this is not basically different from the traditional concentrator approach. Since its presence does not appear a necessity from a communication standpoint, why not do away with it altogether?

DISTRIBUTED HOST

Communications between hosts are not an objective per se. What is actually aimed at are communications between entities, which represent a meaningful set of self-contained correspondents, e.g. processes, files, devices, jobs, subsystems, etc. . . . Since many of these entities are usually housed and activated simultaneously on a single host computer, it is customary to refer to an H-H protocol while what is meant is actually the set of communication rules between corresponding entities in different hosts.

For generality, communication rules tend to be inde-

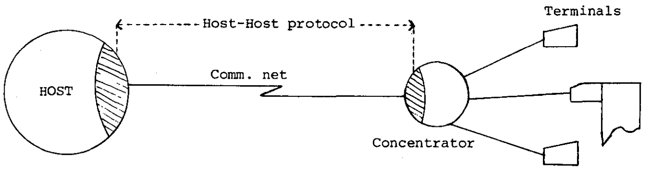


Figure 1—Terminal concentrator

pendent of the nature of the corresponding entities, at least for a class of basic services. Therefore, entities are not specified in the definition of protocols. Instead, there appear anchoring points called *ports,* to which they are bound at communication time.

Although they abide by the same protocol, communications between pairs of ports are logically independent from one another. However, they must share some resources, like an I-O device, a telecommunications package, a communication network. Therefore, protocols must include the information necessary to multiplex shared resources in an orderly manner, so as to preserve logical independence. On the other hand, there is no point in making it a requirement that ports belong, somehow to certain host computers. Whether they happen to share the same computer or not is immaterial as far as communication protocols are concerned. A host address is no more than an information required for the multiplexing of some resources. It does not have to be a particular computer.

For example, in the network structure of Figure 2, the concept of host reduces to a set of transmission lines and packet concentrators, which are a shared resource bearing a host name.

Such a construction may be termed a *distributed host.* One may notice that in this example the technology of packet concentrators is not at stake. They may be programmed mini-computers or hard-wired delay registers. The same concept of distributed host would hold.

It should be clear by now that an H-H protocol on a distributed host cannot be implemented on a particular computer. Furthermore it may well be that nowhere can be found enough computing power to implement a protocol of a few thousands instructions. As a result, only simple and compact protocols are acceptable.

PORT-PORT PROTOCOL

Since a host may not represent a geographically well defined destination, there should be literally no H-H protocol. When two ports communicate, there should not be any logical interference from other ports, hosts, or communications network. In other words, all the machinery interposed between ports should be *transparent.* E.g., protocols used at inner levels should remain invisible. Port-Port protocols are just end-to-end (E-E) protocols.

In order to facilitate communications in a heterogeneous environment of hosts and terminals, it would be desirable to adopt existing or proposed standards. The trouble is that no E-E protocol suitable for networks has yet been advanced in standardization bodies such as ISO or ECMA. HDLC[5] is only valid for wire-like transmission media.

One can vaticinate that HDLC with double numbering scheme will be a de facto standard at some point in the near future, since it is already introduced by IBM in its new products lines. Although HDLC is actually designed only for the control of a physical data link, it may well serve as a starting point for the design of a network E-E protocol. This approach may result from sheer inertia, or mimicry, or IBM pressure on the market. Another reason is that HDLC contains some basic features reasonably close to what might be required for an E-E protocol. Anyhow it seems worth attempting to define an E-E protocol applicable to communications between ports by borrowing from HDLC. At least that may sound as a sensible enough move on the standardization scene.

PROTOCOL TRANSPARENCY

Protocols between ports are only sub-functions invoked by higher level logic for the purpose of transferring information from one domain to another. This transfer action is supposed to remain transparent in terms of information contents. In other words, no alteration should occur. Actually, no physical communication medium can be assumed error-free. Thus mechanisms for error detection and recovery are usually associated with E-E protocols in order to lower the error rate down to a required level. However there remains always some residual probability of error, which may be considered as acceptable, or negligible, for the purpose at hand.

Indeed higher level constructions may contain additional mechanisms which are able to cope with E-E pro-

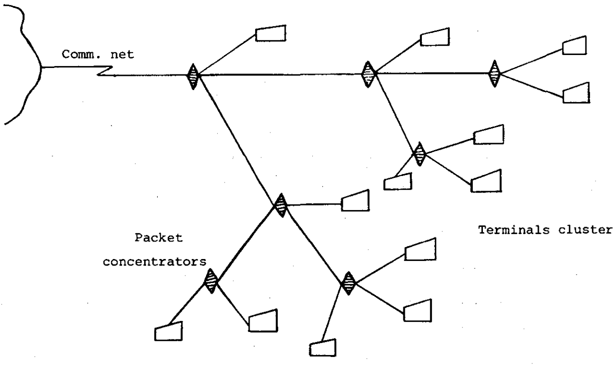


Figure 2—Packet concentrators

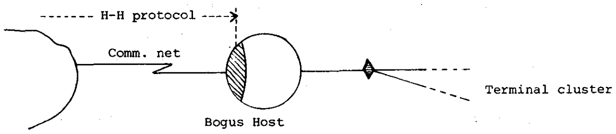


Figure 3—Bogus host

tocol errors, and make them disappear for all practical purposes. In other words, E-E protocols are designed to be normally transparent, except for residual errors, which may even be detected, but not necessarily recovered. What are the nature and the rate of acceptable errors is a matter of economics in distributing properly control functions across levels of protocols.

HIGH LEVEL PROTOCOLS

Transferring information is only a tool toward the achievement of some practical goals, e.g., interaction between process and terminal, or file transfer. These are typical examples of high level functions which are so commonly used in networks that they are worth packaging in some standard form. They also have come to be called protocols in the network literature,[6] such as virtual terminal protocol, or file transfer protocol.

A high level protocol such as file transfer may involve more than two hosts. It is no longer a simple E-E protocol. Actually a high level protocol is a set of rules defining the working of a distributed machine, which is designed for handling a particular application. One may notice that this definition applies to E-E protocols as well.

In order to perform its task a distributed machine must accept commands, input or output data, and transfer information between its components. These exchanges may carry data, or commands and state variables necessary for the synchronization of the whole machine.

While data can be altered intentionally, as part of the machine task, (e.g. data conversion or reformatting), communication mechanisms between the distributed components should be ideally transparent. Thus E-E protocols may be used as building blocks to carry out any information transfer required by a high level protocol. This makes for strong incentives to define general and efficient E-E protocols. Otherwise the piling up of layers of protocols would ultimately result in excessive overhead.

PROTOCOL NESTING

A unit of information, say a message, exchanged between a terminal and a process may be restricted in length by mutual agreement. But this is not necessarily short enough to comply with the characteristics of a communication network, which may use smaller units for various reasons of cost-effectiveness. Therefore messages to be transmitted must be fragmented, and the fragments must be reassembled at the destination.

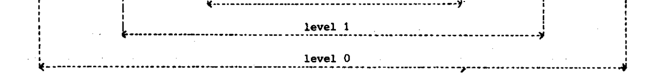


Figure 4—Protocol nesting

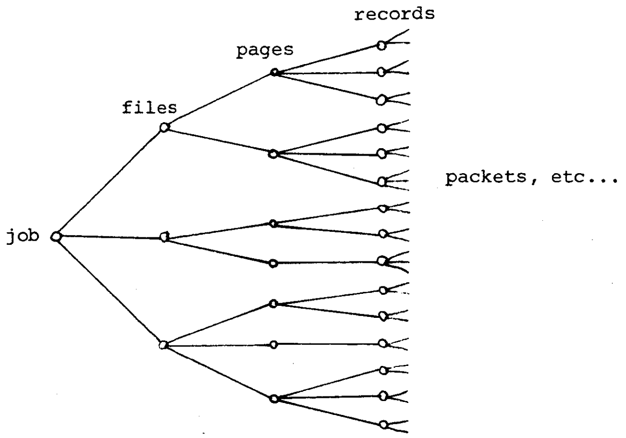


Figure 5—Protocol tree

Similarly, transferring a file cannot usually be accomplished by just sending one message. The file must be fragmented in blocks, pages, records, which are sent individually, and put back together onto the destination file store.

Sending a job to a distant computer may involve the transfer of a set of files. More generally transferring a unit of structured data requires breaking it into a set of pieces, which are again broken down repeatedly until one reaches a level of fragmentation suitable for physical transfer, (Figure 4). This scheme is general, and does not make any assumption about the physical or the logical structure of data.

At each level of nesting there appear similar functions:

- break logical unit into fragments
- transfer fragments separately
- put fragments back into logical unit

This is again repeated, as each fragment becomes a logical unit and is broken down in sub-fragments, etc. . . . up until no more fragmentation is necessary. A way of representing this process is traversing a tree structure, (Figure 5).

A first idea that comes immediately to the mind is that transferring information could well be a recursive application of the same protocol.

Let us call S (N,F) the function called upon at the Nth level of fragmentation for sending the fragment F. The function S (N,F) may be expressed as follows:

```
sender: procedure S (N,F)
        begin if fragmentation required then
                begin make fragments f (1) . . . (p);
                      for i: = 1 step 1 until p do S (N+1,f(i))
                end
          else send F
        end
```

The corresponding function R (N,F) at the other end might be expressed as follows:

```
receiver: procedure $R$ $(N,F)$
begin if fragmentation then
      while fragment missing until time-out do
            begin $R$ $(N+1,f)$;
         if $f$ good then place $f$ in $F$
         else drop $f$
      end
   else receive $F$;
   if $F$ bad or time-out then report $F$ bad
end
```

These algorithms are obviously a first approximation. They contain a limited set of functions necessary to carry out communications. It may now be appropriate to examine what differences or additional requirements might appear in practice.

Error recovery

The previous algorithms detect errors and report them to the next higher level of control. Ultimately the highest level gets the report and starts all over again. But this may not be satisfactory in terms of efficiency. Transit delays and possibly transmission overhead are reduced when error recovery is performed at lower levels, on small fragments.

E.g., one might introduce additional functions whereby the receiver sends back an acknowledgment for each good or duplicate fragment received. At the other end, a repetition process sends again any fragment not acknowledged within a certain delay. As can be predicted, this error control scheme induces overhead of its own in transmission (ACK information) and in processing.

Furthermore, handling error recovery at a particular level does not make it error free for all higher levels, because it cannot be guaranteed to be always successful. Consequently, the highest level must always be prepared to get an error report. Recovery at lower levels can only reduce the error rate down to an acceptable figure.

There is no simple criterion to help determine at which level error recovery would be the most effective. First, error control overhead in transmission, processing, or buffering depends on the many variants in acknowledgment schemes. Second, error and traffic patterns, user constraints, system characteristics, are also major factors bearing on error control effectiveness.

Therefore, one may say that error recovery should appear at some levels of protocols, depending on the environment. A convenient way is to make it an option, both in time, and at a particular level.

Flow control

Resources necessary for transferring information must be made available, on the spot, or by prior reservation. Allocation schemes may be desirable to prevent traffic unstability and interferences between independent correspondents.

Again flow control adds up its own overhead, and it is likely not desirable to place it at all times at all levels. This is also a feature which one would like to turn on and off at a particular level, depending on the environment.

Fragment identification

A noticeable difference in handling fragments at the receiver end derives from the sequence of arrivals. When the sending order is normally maintained by the communication medium, any fragment can be identified by its order of arrival. Since errors may occur, fragments carry a sequence number, and any fragment arriving out of sequence is an indication of error. At this point recovery may take over, at some level of protocol.

On the other hand, when fragments may normally arrive out of sequence, they should not be rejected, except if this occurrence is practically negligible. Thus some buffering is needed to park temporarily early fragments waiting for logical predecessors. In the algorithms given previously, there is no assumption about the arrival sequence, as long as all fragments belong to the same next higher level unit. Otherwise, two alternatives may be considered:

(a) fragments of the next unit are rejected. They may be kept from being sent by synchronizing the next higher level of protocol at unit level.
(b) fragments belonging to a future unit are just stored until the corresponding instance of the protocol is activated.

It is clear that fragments must be labeled so that such an identification can take place.

Other differences

The semantics of the algorithms differ depending on the physical and logical structure of fragments. E.g., reassembling pieces of a file on disk is not the same as for a message in core memory. Reporting a failure may take different forms. Time-out's are adjustable parameters. System protection and access methods may introduce quite a number of differences in the practical handling of fragments.

For all sorts of reasons mentioned previously, it may not be practical to use identical routines or algorithms to implement the various levels of protocols. However, similarities are substantial enough to warrant some effort toward a general scheme applicable at all levels.

Control information

In order to handle each fragment properly, some additional information is necessary for fragment identification and to allow each end of the protocol to work in synchronization. This is usually packed into a header preceding the text or the descriptor of the fragment proper.

When nesting protocols a classical method is that each level wraps every fragment in its own envelop. If further fragmentation is to occur, the complete fragment (header and text) is broken into smaller pieces, each of which receives a new header sticked by the nested protocol. This works of course, but each level of header increases the amount of transmission overhead. If each level of fragmentation produces statistically a large number of smaller fragments, then headers at intermediate levels contribute a relatively small part of the total overhead. But nesting does not always produce many fragments, since it may be required only to pass through a particular layer of the system hierarchy. Then each level of wrapping duplicates to a large extent control information in every nested header.

In compensation for overhead there is the advantage of allowing the design of each level independently. For lower levels, fragments are just unrelated pieces of information. This approach is indeed desirable when it is essential to avoid unnecessary coupling between layers of systems, e.g., when some layers are shared by unrelated users.

On the other hand when system-wide consistency is desirable for efficiency, and in the domain of realistic objectives, one may attempt to define a common set of control information intended to be used by every layer of protocols. This may reduce effort in design, help standardize well defined protocols, and ease out error recovery, since every fragment may be directly related to a logical higher level unit of information.

NESTED PROTOCOL CONTROL

When implementing a full-fledged protocol at some level of communications, it is possible to identify two classes of control information exchanged between sender and receiver. One class is associated with the direct handling of the text part of messages, in order to make sure it is correct, and to locate it in relationship with other text parts. Another class pertains to the specific mechanisms intended for the synchronization of the two ends of the protocol.

In a nested environment, the first class may be called global variables, since the text is independent of the protocol level. The second class constitutes local variables which are part of the execution context of the protocol. They should not normally appear at other levels, except for those variables used as run time arguments when control passes up and down across levels of nesting.

Actually, one may often identify a third class, such as "piggy-back" information. But this is in essence independent of the protocol. Only messages are used as ready to go containers for hitch-hiking information that belongs to another protocol, i.e., the sender-receiver pair dealing with traffic in the reverse direction.

In the following we shall examine what control information could make up a common header at all levels of nesting. The objective is that each level of protocol should only modify certain fields of a fragment header, rather than adding its own. When fragmentation occurs, each sub-fragment receives a similar header, with only changes in certain fields. There are also suggestions for field sizes.

P-level (4 bits)

This is a number indicating the nested level of the protocol in charge of processing this message. (Higher numbers are lower levels of protocols; they may be considered as negative if one wants to keep consistency in terminology.)

U-NR (7 bits)

This is the identification number of the logical unit being fragmented or reassembled at this level of protocol. It is also a fragment number passed as argument between this level and the next higher. Its use is to segregate fragments of other units arriving out of order.

F-NR (7 bits)

This is the fragment identification number of the text part carried within this message. It becomes a U-NR passed to the next level down when further fragmentation is required. It may be used circularly when the number of fragments exceed the maximum value allowed by the field size. But this assumes that chances are nil that a message gets out of sequence by such a gap. If this cannot be ascertained, the number of fragments is limited to the maximum value for F-NR.

F-TOT (8 bits)

This is the total number of fragments created or expected at this level of protocol. It is intended for resource reservation.

E-level (4 bits)

This is the level of the protocol handling the unit of which this message is the last fragment. In the protocol tree, it is a pointer back to a higher level in a traversing process.

ALT (1 bit)

This is an alternation signal used for various purposes such as checkpoints, or conversational interactions. It corresponds to the P/F bit of HDLC procedures.[5] It pertains to E-LEVEL, if any, otherwise to P-LEVEL, in the protocol hierarchy.

Table I—Fragmentation Scheme

	P-LEVEL	U-NR	F-NR	F-TOT	E-LEVEL
1	a	b	c	d	e
2	a+1	c	1	f	
3	a+1	c	2	f	
4	a+1	c	3	f	
5	...	...	...	...	
6	a+1	c	b	d	e
7	a	b	c	d	e

COM (4 bits)

This is a command field allowing for a variety of interpretations of message formats.

ACK-ALL (16 bits)

This is piggy-back information intended to carry acknowledgment and allocation of messages for the reverse traffic. No assumption is made in this paper about the particular scheme used. The ACK-ALL field may be optional. It can contain a level number so that it be forwarded to the proper level of protocol, independently of the P-LEVEL of this message.

Coding and packing these fields toward an efficient format is left for further study.

Principles of operation

A simple example is likely the best way to explain out how transfer mechanisms work out.

Let us assume that a particular level of protocol has to fragment and transfer a unit of information with the parameters indicated in (Table I), row 1. The number of fragments generated will be **f**. Thus, it produces a series of messages with headers indicated in rows 2-6. The last fragment identified by the presence of E-LEVEL carries the parameters intended for the protocol at level **e**, which is to reconstitute a complete unit. Thus the receiver part of the protocol at level **a** is able to piece together all the parameters of its message, check it for validity, and make up a unit for its next higher level. If there is no E-LEVEL indicator in row 1, then the last message in row 6 will carry the value a for E-LEVEL.

There is no need for separate initialization, except at the highest level. Error and flow control may be exercised independently at any level on an optional basis. It is possible from the highest level to turn these options on and off, through appropriate commands directed to the proper level. A negotiation protocol may be introduced, so that the implementation of options be also optional. Due to these facilities, the ultimate user is in a position to select an efficient set of characteristics throughout all levels.

INTER-NETWORKING

The inter-connection of separate networks raises a number of problems which have been addressed in previous papers.[7,8] One particular issue revolves around the maximum message size accepted by each network. A first approach is to assume a minimum size which every network would be able to carry. Thus inter-network traffic would use a maximum message size commonly agreed as a standard. This is typically what has been defined as the "datagram" service within the CCITT. A comprehensive inter-connection scheme based on this approach can be found in Reference 8.

Another approach assumes that some networks will offer larger message sizes than others, and that inter-network traffic should take advantage of larger sizes as much as possible. To that effect networks would be interconnected via gateways which would fragment messages further whenever they happen to be too large for the network downstream. A controversial implication of that scheme is that a network accepting only small messages would induce additional overhead in all successively traversed networks including the final receiver host. Also the fragmentation scheme in gateways would have to match tightly the E-E protocol between hosts.

It does not appear possible to do away with these drawbacks, which are intrinsic to any fragmentation scheme located within communication networks. To the extent that the implications of inter-network fragmentation are acceptable, then the protocol presented in this paper applies perfectly to the situation.

Indeed, no particular assumption has been made about the mechanisms transferring messages across protocol level boundaries. As a possibility, successive levels of sender protocols may be located in the gateways of several interconnected networks. Whenever necessary a message passing through a gateway will be broken down into several fragments. At the final destination, all matching levels of receiver protocols coalesce into a single procedure putting back together all arriving fragments through a recursive application of the protocol at the level indicated in each message header.

If at all useful, error recovery and flow control can be exercised between the final destination and any of the sender levels, including the traversed gateways. This may improve efficiency by reducing repetitions, and smoothing out traffic.

Furthermore, if it turns out that for a certain destination all messages created at a certain level of fragmentation must actually travel through a single gateway, it may well reassemble pieces into higher level-units, rather than let them travel individually. This would reduce overhead downstream. Such a case is typical of networks requiring an initial virtual call set up. Then all fragments are constrained to leave the network through the same gateway. This is indeed a rare case where the clumsiness of virtual calls is put to an advantage.

STANDARDIZATION

There is not yet a final consensus for an HDLC standard within ISO. However proposals tend to stabilize around an independent double numbering scheme, with provision for address and control field extensions. As such HDLC would not meet the requirements brought up in this paper. Nevertheless the principles introduced by HDLC are reasonably close, viz:

- independent double numbering
- time delay between sending and acknowledgment
- data transparency

Field extensions and additional modes of operation could turn HDLC into an acceptable multi-level transfer protocol such as described here. Furthermore, a simple point-to-point physical data link is just a degenerated network, and could be under control of a network protocol. Thus at some point in the future, a general network protocol, with negotiable options, might be suitable for most kinds of information transfer, regardless of data structure, message size, and communication medium.

The recursive approach, combined with implementation options, should lead to simple and compact microprogrammed protocols integrated within intelligent terminals. On the other hand, if levels of protocols are distributed within a system hierarchy, each level may be implemented with a particular optimization in mind, while the complete structure remains consistent and easily understood.

REFERENCES

1. Crocker, S., A. McKenzie, J. Postel, *Host-Host Protocol for the Arpa Network.* NIC 8246, January 1972, 37 p.
2. Zimmermann, H. and M. Elie, *Transport Protocol: Standard Host-Host Protocol for Heterogeneous Computer Networks,* IFIP WG6.1, Doc. 61, April 1974, 31 p.
3. Farber, D. J., K. C. Larson, *The System Architecture of the Distributed Computer System,* The communications system Symposium on computer-communications networks and teletraffic, April 1972, pp. 21-27, Polytechnic Institute of Brooklyn.
4. Denjean, F., *Connexion de Terminaux à un Réseau de Commutation de Paquets,* Internat. meeting on mini-computers and data communications, Liège, January 1975, 8 p.
5. ISO/TC 97/SC6—*High Level Data Link Control Procedures,* Doc. 1005, October 1974, 54 p.
6. Crocker, S. D., J. F. Heafner, R. H. Metcalfe, J. B. Postel, "Function Oriented Protocols for the Arpa Computer Network," *SJCC,* June 1972, pp. 271-279.
7. Cerf, V. G. and R. E. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Transac. Comm.,* May 74, pp. 637-648.
8. Pouzin L., "A Proposal for Interconnecting Packet Switching Networks," *Eurocomp,* Brunel Univ., May 1974, pp. 1023-1036.

Interaction monitors in a distributed system

by RAJIV MALHOTRA
Burroughs Corporation
Detroit, Michigan

INTRODUCTION

By utilizing the parallelism and redundancy inherent in a network, a set of interacting processes could be made more effective. But this should not require the user to know the network details: the network should be transparent enough to allow him to treat it as a single entity rather than as a collection of computers. The concept of relocatable programs within a machine could be generalized to allow processes to be relocatable anywhere within the network. Then, several mappings could exist from a given set of processes to the set of computers in the network, with the final results being independent of the mapping used.

As a step toward such generality, one needs adequate constructs to conceptualize and represent the interactions among processes independently of their geographical locations. It is, therefore, desirable to have some "atoms" of interaction that are equally vaid for processes in separate computers and for processes in the same computer. The implementation philosophy for constructs which perform this interaction should be such as to allow easy additions of further constructs.

There are two aspects of a set of asynchronous processes in a network: the dynamic creation and deletion of processes and the interaction among these processes once they have started. In this paper, we deal only with the second of these aspects.

Existing IPC (interprocess communication) schemes merely allow messages to be transferred between processes.[1,4,6,16] Although this facility is necessary, it is not always a convenient programming tool. Accordingly, some higher-level (or function-oriented) protocols have been introduced to provide specific types of interfaces, and thereby isolate the user from the mechanics of communication.[5]

In parallel with this approach, we suggest that each type of interaction among a set of processes should be controlled and coordinated by a special process called a monitor. Some synchronization monitors are discussed as examples in the appendix. Other kinds of monitors may be introduced in a similar fashion. The notion of synchronization is introduced below.

Synchronization is any timing constraint placed on interactions among concurrent processes. In a typical situation, one has a set of interacting processes whose speed ratios are not predictable; yet constraints exist on the relative sequencing of certain operations performed by them. At one extreme, such a set of processes could have almost unrestricted concurrency. At the other extreme, the processes could require synchronization to run "in step" most of the time. In many applications, there are "units of action" (often called transactions) that are allowed to interleave arbitrarily but operations within such units have to be synchronized.[2,8,9] This creates a degree of serialism in a set of concurrent processes.

In the following sections, references will be made to some specific synchronization constructs found in the appendix. Their syntax is similar to that suggested in the literature for conventional operation systems although their effects span across machine boundaries. An open problem in the design of interaction constructs is to allow compilers to check for most time-dependent errors.

The remainder of this paper contains a general description of monitors and their implementation in a distributed system.

MONITORS

A set of processes that wish to interact may do so by referring to shared objects. A *shared object* is an entity that may be known to more than one process. Each process using a given shared object must declare it with the attribute "SHARED".

Just as in the case of ordinary objects local to a given program, shared objects are of various *types*. The type of a shared object determines a valid set of operations on that subject. Examples of types of shared objects are: file, semaphore, event and conditional region. (Some of these shared objects are discussed in the appendix.) One shared object could be appropriate for a particular kind of interaction but not for another.

A shared object may be either a constant or a variable. For instance, if **s** is a constant of type semaphore and **x** is a variable of type semaphore, PROCURE (**s**) will have the same effect as:

x: = **s** ; PROCURE (**x**);

To manage a shared object, there is a special kind of process called a *monitor*. Each time a shared object is

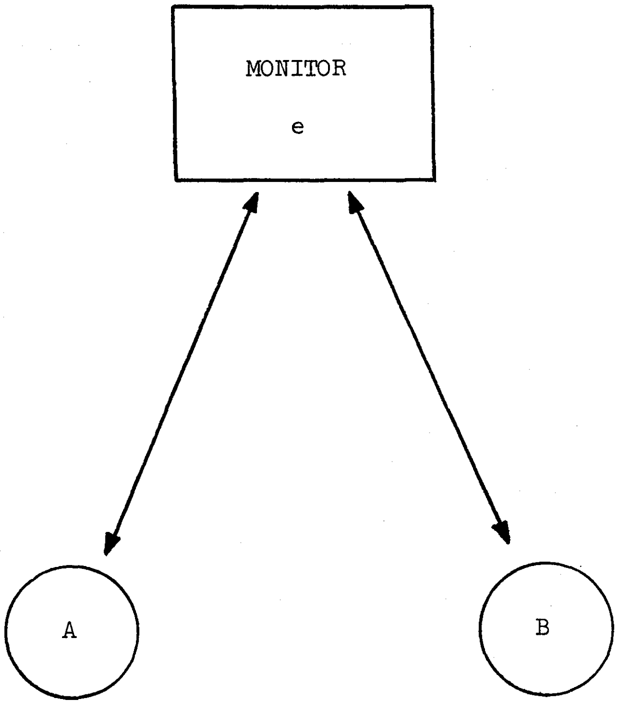


Figure 1—Processes **A** and **B** referencing a shared object **e**

referenced, the local system software sends a message to the monitor of the shared object. All such messages (called *requests*) to a given monitor are queued. When the monitor services a request in its queue, it sends responses back to one or more nodes. This transfer of messages is totally invisible to the interacting processes.

In our theoretical model, there is a one-to-one correspondence between shared objects and monitors, and hence, these terms shall be used interchangeably. In practice, the work of many monitors could be accomplished in a single process.

Figure 1 shows processes **A** and **B** referencing a shared object **e**. The monitor for **e** is located at one of the nodes in the network. The management of **e** is invisible to **A** and **B**. If **e** were an event, then each time a process executed an operation to WAIT for **e**, that process would get suspended and a message would be sent to the monitor of **e** (by the system software). Monitor **e** would maintain a list of all such processes waiting for **e**, and when any process executed a CAUSE operation on **e**, monitor **e** would send messages to certain nodes asking the system software that all the process waiting for **e** be restarted.

There is a distinction between real and virtual times. Real time is made available by a hardware clock. Virtual time is relative to a given process and corresponds to state changes of that process. Thus, if a particular process remains suspended for a while, then its virtual time does not elapse although the real time continues progressing. Each operation on a shared object is instantaneous in the virtual time of the process but will necessarily have a delay in real time. For instance, a PROCURE operation could involve a long period of real time but the state of the process remains unchanged.

Even though any single operation is instantaneous in virtual time, the overall interaction between a process and its environment could be such that while the interaction is in progress the process is doing some other work. Such constructs allow a process to interleave several interactions in virtual time. For instance, if a process wishes to procure a semaphore, it could periodically do a TRYTOPROCURE operation and on failure spend its time doing something else. On the other hand, if a process has no other work to do and is waiting for a certain condition, it is more efficient for it to get suspended for the interim period. (This is known as the "non-busy" form of interaction.) The choice of how to interact would be based upon the extent to which a process depends on the interaction. A "tightly coupled" set of processes would prefer this non-busy form of interaction.

In the next section matters related to an implementation of monitors are discussed.

SOFTWARE STRUCTURE

To implement operations on a shared object without explicit message transfers, one needs some intermediate system software, called the *interaction system,* in each computer. Specifically, if a process executes an operation on a shared object, the local computer's interaction system receives control. The process itself may be suspended. The interaction system searches its tables to find the corresponding monitor for this shared object, and sends a message to it requesting appropriate action or status information. Since the interaction system must be able to si-

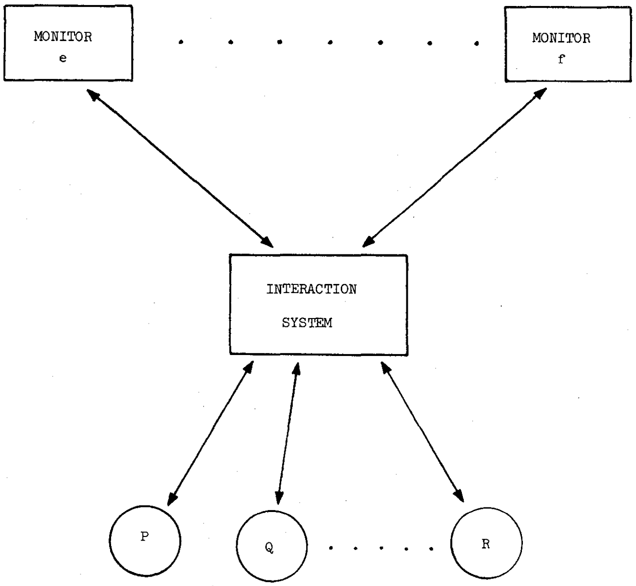


Figure 2—Interaction system of one computer interfacing between all its local processes and all the network's monitors

multaneously handle many operations, it makes entries in its tables to save the state of each operation in progress. When a reply from a monitor arrives, the interaction system interrogates its tables to determine the processes that are affected by this reply. For instance, if a message arrives from a monitor indicating that a particular event has been caused then all local processes that are waiting on that event will be started. Figure 2 shows the interaction system of a computer interfacing to the local processes on one side and the relevant monitors of the network on the other.

The next matter to be discussed is how the interaction system communicates with the various monitors located in different parts of the network. We assume that the distributed network already has an IPC scheme. At least three such schemes are known.[4,6,16,1] Basically, an IPC capability allows any two processes in the network to establish communication under mutual agreement and transfer messages. The various interaction systems and monitors in the network would use this IPC. Figure 3 shows two processes using this facility. From a logical viewpoint, a communication path or channel is established for the message flow. Depending upon the IPC scheme being used, the processes may have to know each other's names, port numbers, socket numbers, etc.[1] This requires that the interaction system should know which monitors are required by its local processes and what the addressing details of these monitors are. Some IPC schemes would require that a "connection" be established between an interaction system and the monitors. Since the interaction system would have a limited number of logical channels at its disposal, it would allocate these channels as a resource to the individual operations currently active, based on demand and priority.

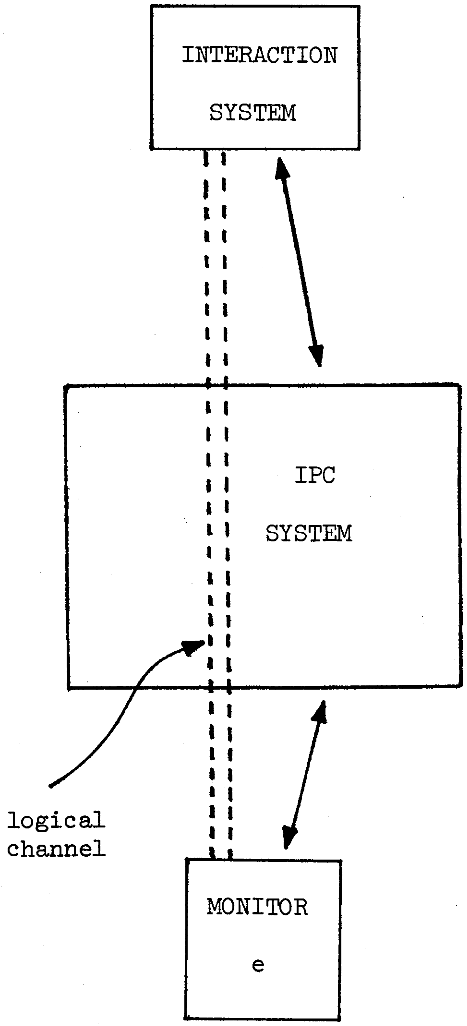


Figure 3—An interaction system and a monitor communicating via the network IPC

NAME SPACES

It should be possible for any set of processes in the network to interact by referring to the same shared object. On the other hand, two totally independent processes belonging to different users should not interfere with each other just because each of them declared a shared object with the same name. The basic requirement is that processes should interact if and only if they wish to do so. In this section, the general problem of managing the monitors in a network is discussed.

Note, if an object is declared as being local to a program (although it is shared by the various dependent processes compiled as parts of the same program), then this declaration is invisible outside of this program. The problem of name clashes arises only in the case of objects declared as being shared globally, and hence potentially accessible to separately compiled programs running anywhere in the network. It shall be assumed that the shared objects discussed below are global ones.

A new monitor is created when a user declares a shared object with the attribute "NEW". This may be done in a job control language or as a system command. One may name any node of the network as the location of this monitor, or else a default location will be provided. It is important to note that the interacting processes are unaware of a monitor's location; however, its location may be important for efficiency considerations. Any user may maintain his own supply of monitors by creating and deleting them dynamically.

A naming scheme is described below that allows each monitor in the network to be identified uniquely. The name by which a monitor is known throughout the network is called its external name. A process may refer to it using an internal name (relative to that process) which may be equated to the external name at run time. This is analogous to equating file names at run time.

Every user in the network has a unique identification and all processes and monitors created by him are tagged with this identification. For a given user, there is a distinct set of directories, one per type of shared object. A shared object occurs in exactly one directory. Effectively, the set of all user directories for a given type of shared object constitute a network-wide name space for shared objects of that type. When a process refers to a shared object, it need not specify its type since this can be found by the compiler from the context.

Entries in a directory contain information such as the physical locations of monitors and the access rights to them. When a new process is created, the local interaction system searches the network name spaces to find the locations of all the monitors required by it. Another approach is to perform the search for a monitor the first time it is referenced. The important point is that a search is not necessary each time a given shared object is referenced.

The owner of a shared object may restrict access to it based on a password. He could also limit the operations that another user may perform on it. In the case of an event, for example, he may allow other users to wait on it but not cause it or delete it. This enables the following to be accomplished:

(1) Any user may set up his own "logical network" of processes that interact using shared objects belonging to him. He may dynamically alter this set of shared objects.
(2) Two separate users will not accidentally interfere due to name clashes between their shared objects, since each user has his own distinct directories.
(3) Under mutual agreement, processes of different users may interact to form larger logical networks. Monitors belonging to one user may be accessed by others subject to constraints imposed by the owner.

To make it convenient to use a shared object's external name, one may implement "short hand" naming conventions. Since the name space of a given type of object is a set of user directories, it forms a tree and any element in it may be referenced by its pathname. A user could be allowed to establish a reference point in the name tree and assume all pathnames to be relative to it. The reference point may be altered dynamically. Basically, this enables one to set up a "working directory". A working directory could also be specified as a set of sub-trees of the name tree and may include shared objects of other users. An appropriate default working directory is the user's own directory. To make the working directory concept more powerful, an installation may maintain a user profile that defines this user's default working directory. A user would be able to change his profile. This subject is further explored in Reference 14.

CONCLUSION

A fundamental premise of this paper is that a network programmer would benefit from the availability of programming constructs that allow him to be unconcerned with the "lower-level" communication in the network. This paper discusses an approach for accomplishing this and uses conventional synchronization constructs as examples.[2,3,7,8,13] With the use of such constructs, the network appears to be a single, large, multiprocessor system.

A major goal to be accomplished is to develop network operating systems that make resource-sharing more effective.[6,14,15] Programming constructs which are effective network-wide would be helpful not only to users but also in the implementation of a network operating system.

Several problems remain that need to be addressed in future work:

(1) Better interaction constructs must be invented that make it easier to prove program correctness.[9]
(2) If some of the interacting processes should fail, those remaining might be adversely affected. Research is needed to prevent individual failures from propagating throughout the network. Perhaps, the various interaction systems should perform periodic "handshakes" with each other to detect node failures, and should continually interrogate the status of processes in their own respective nodes to detect process failures. Upon failure detection, a recovery would have to be coordinated.
(3) Any resource-allocation system is confronted with the problem of deadlocks. Conventional algorithms for detecting and handling deadlocks[11] need to be adapted for distributed systems. Perhaps, a special monitor for the entire network could interrogate all other monitors to detect deadlocks. Either it could run periodically or other monitors could trigger it when suspecting a problem.

Within the environment proposed in this paper, a system designer would have to keep the density of interaction among processes low so that the message traffic in the network does not become unbearable.

It would be desirable to make the interaction constructs extensible. This would allow a user to write his own monitor and define appropriate operations for it.

ACKNOWLEDGMENTS

The author wishes to acknowledge the assistance of Mr. Joe Tuharsky for editing the manuscript and Ms. Sharon Bischoff for typing it.

REFERENCES

1. Akkoyunlu, E., A. Bernstein and R. Schantz, "Interprocess Communication Facilities for Network Operation Systems," *COMPUTER*, June 1974.
2. Brinch Hansen, P., *Operating System Principles*, Prentice-Hall, 1973.
3. Burroughs Corporation, "B6700 System Miscellanea," Form No. 5000367, April 1974.
4. Carr, S. C., S. D. Crocker and V. G. Cerf, "HOST-HOST Communication Protocol in the ARPA Network," *SJCC*, 1970.
5. Crocker, S. D., J. F. Heafner, R. M. Metcalfe and J. B. Postel,

"Function-Oriented Protocols for the ARPA Computer Network," *SJCC*, 1972.

6. Farber, D. J. and K. C. Larson, "The System Architecture of the Distributed Computer System—The Communication System," *Symposium on Computer Networks*, The Polytechnic Institute of Brooklyn, 1972.
7. Fisher, D. A., *Control Structures for Programming Languages*, Ph.D. Dissertation, Carnegie-Mellon University, 1970.
8. Haberman, A. N., "Synchronization of Communicating Processes," *CACM*, March 1972.
9. Hoare, C.A.R., "Towards a Theory of Parallel Programming," *Operating Systems Techniques*, Academic Press, 1972.
10. Malhotra, R., *Process Synchronization in a Computer Network*, privately circulated.
11. Murphy, J. E., "Resource Allocation with Interlock Detection in a Multi-Task System," *FJCC*, 1968.
12. Somia, Monique, "Synchronization Problems in a Computer Network," *International Computing Symposium*, 1973.
13. Spier, M. J., "Process Communication Prerequisites or the IPC-Setup Revisited," *1973 Sagamore Computer Conference on Parallel Processing*.
14. Thomas, R. H., "A Resource Sharing Executive for the ARPANET," *NCC*, 1973.
15. Thomas, R. H., and D. A. Henderson, "McROSS—A Multi-Computer Programming System," *SJCC*, 1972.
16. Walden, D. C., "A System for Interprocess Communication in a Resource Sharing Computer Network," *CACM*, April 1972.

APPENDIX

To illustrate the notion of a shared object, some examples are discussed in this appendix. These are by no means exhaustive. The reader is referred to References 2, 3, 8, 9, 13 for further study.

Semaphores

A semaphore may be used to represent a resource that has to be reserved before it can be used. If **s** is a semaphore, PROCEDURE(**s**) suspends the process until **s** is available and then the process is restarted and "owns" **s**. The PROCURE construct causes the local interaction system to send a request to the appropriate monitor. When a process wishes to relinquish control over **s**, it executes a RELEASE(**s**). The choice of which process wakes up in case several are wanting to procure **s** is undefined and it is assumed that this does not matter.

The integer function PROCURE (**s**, **t**, **u**, **v**, . . .) has the "side-effect" of procuring exactly one of the semaphores in the argument list. The value returned denotes the relative position of the semaphore that was procured. A suitable use of this would be in a case statement as follows:

```
CASE PROCURE (s, t, u, v);
           :
END CASE;
```

In order to specify a time limit on the procurement operation, the integer function PROCURE (<time limit>, **s**, **t**, **u**, **v**, . . .) may be used. If a timeout occurs, the value returned is one. The integer function TRYTOPROCURE (**s**, **t**, **u**, **v**, . . .) returns a zero if none of the specified semaphores is immediately available; it is similar to the PROCURE construct with the time limit set to zero.

In a general situation, a certain maximum number of processes may be allowed to own a particular semaphore at any time. This maximum number is called the *capacity* of the given semaphore and may be defined by its owner. At any time, SUPPLY(**s**) returns the number of copies of **s** that are still available; QUEUESIZE(**s**) returns the number of processes contending for **s**. These functions could help a programmer in scheduling his "bids" for semaphores.

Events

An event is a type of synchronization object suitable to transmit timing signals to another process or to indicate that a certain condition has been satisfied. At any given time, an event **e** is either in the set or reset state. When a process does a WAIT(**e**) and **e** is already reset, it gets suspended until **e** gets set. A message is automatically sent to monitor **e** who is responsible for informing all waiting processes when it's time to wake up. A process may wait on several events, including timeout, using the function WAIT (<time limit>, **e**, **f**, . . .) which returns an integer value indicating which of the events occurred. The time limit is optional.

Event **e** is set (occurs) when a process does a CAUSE(**e**). Once set, it remains set until a RESET(**e**) is executed by one or the processes, and as long as it remains set no process will have to wait on it. The operation CAUSEANDRESET(**e**) will wake up all processes currently waiting on **e** and then immediately reset **e**.

An interrupt **I** is a procedure that is ATTACHed to an event and will be triggered whenever that event is caused. When the procedure **I** is finished, control will be returned to the point where the process was interrupted. Several programs may attach one of their procedures to the same event and all of these will be triggered when the event occurs. Since these are "soft" interrupts, there is an unpredictable lag (in both real and virtual times) before the interrupt is actually felt. The monitor of an event keeps track of which interrupts are attached to it. A process may dynamically ATTACH or DETACH an interrupt from an event.

Conditional regions

Although semaphores and events allow a variety of interactions to be programmed, researchers are interested in developing constructs that enforce a better program structure. In sequential programming, a lot of work has been done to expose the trade-offs between "tight", unstructured programs and structured programs with provable properties. Similarly, in concurrent programming, very highly asynchronous processes tend to be unstructured and often have non-deterministic behavior. Interaction constructs have been proposed that would discipline the nature of asynchronous processing and help to structure systems with more provable properties.

One of the shared objects proposed for this is called a *region*. If **r** is a region then consider the following code:

```
WITH r DO C END
```

Out of all processes that try to execute such a block, only one is allowed to "own" **r** and enter **C** at a given time. This way of implementing mutual exclusion allows for better compile-time checking than is possible with the use of semaphores. (A semaphore could be procured without being released, a condition that cannot be checked by a compiler.)

A generalization of this is known as a *conditional region*. In this case, one may specify a condition involving **r** that must be satisfied in order to own **r**. For instance,

```
WITH r WHEN P(r) DO C END
```

In the above example, **C** will be executed only if P(**r**) is true and if no other process owns region **r**. Here, **r** may be of type Boolean, integer, character, etc., as declared. The compiler checks that **r** is not referenced without being owned.

A more flexible approach might be to associate a conditional region with an event and disallow a process to own that region until the corresponding event has occurred.

Although conditional regions are of theoretical interest, they have not been implemented and, hence, there is not enough experience available on their usefulness.

Area Director:
Vaughn Alexander
Texas Medical Foundation
Austin, Texas

Medical and health care computing

The '75 NCC will provide an all-day forum during which physicians and computer specialists will examine mutual problems and concerns. Emphasis will be on medical and health care users' needs. Problems, applications, and the future of medical computing will be assessed by physicians active in medical computing plus authorities from the information processing field.

The day will begin with a discussion of past and present problems in medical information systems, followed by a presentation of medical applications systems. The first afternoon session will address the emerging area of modular medical applications on small computers. The final session will be a discussion of the future of medical computing, including the direction of graduate education in the field.

Information processing needs and practices of clinical investigators—Survey results

by NORMAN A. PALLEY and GABRIEL F. GRONER
The Rand Corporation
Santa Monica, California

INTRODUCTION

Although medical researchers were among the first to pursue the promise of computerization, clinical investigators, namely physicians who conduct research into the biology of human subjects and who evaluate the efficacy of new therapeutic measures, have not shared even modestly in the benefits so far available. With the expectation that improved information processing techniques and facilities can improve the quality and efficiency of clinical research and ultimately patient care, The Division of Research Resources (DRR) of the National Institutes of Health (NIH) has been sponsoring a scientific inquiry (called CLINFO) aimed at first developing a detailed understanding of the information processing needs and current practices of clinical investigators and then developing methods for generally and economically alleviating some of the most important needs. The CLINFO investigation team consists of clinical investigators (T. Graham Christopher, M.D. at the University of Washington, Arthur W. Nunnery, M.D. at the University of Oklahoma and Howard K. Thompson, Jr., M.D. at the Baylor College of Medicine), the authors and other information scientists at The Rand Corporation, and William R. Baker, Jr., the CLINFO Project Officer, and other staff members of the DRR.

We have taken the approach that before we can effectively recommend or develop computer-based aids for clinical investigators, our entire team must develop a common deep and realistic understanding of clinical investigators' backgrounds, working environments, practices, problems and expectations as well as a similar understanding of the costs, capabilities and reliabilities of present and future computer hardware and software. To do this, we have informally discussed clinical research processes and a broad set of problem areas with many investigators and have conducted a formal survey[1] to collect additional objective data. Over the next three years we will develop a prototype minicomputer-based clinical-research data management and analysis system[2] which we will evaluate in a few research centers.[3] Only after we have tested a set of hardware, software, personnel and operating-procedure specifications, would we be prepared to recommend the widespread utilization of such a system.

We began our investigation by observing and informally interviewing more than 100 personnel at over a dozen clinical research sites. As a result, we identified the following as areas that should be considered further: research-data storage, retrieval and analysis; research center administration; research protocol generation; report writing; and locating potential subjects and tracking outpatients.

After carefully examining tentative approaches to these problems, we hypothesized that present computer technology could have substantial impact on the clinical research community if addressed to the data management and analysis areas for the following reasons:

- Good data management is required to ensure that complete, accurate sets of data are collected, that data are available for timely decision-making, and that information is accessible in a form suitable for analysis both during and following an experiment.
- An experiment is of little value if the data collected are not thoroughly analyzed and interpreted.
- Investigators often do not have the time, background, facilities, and staff for proper data analysis.

Furthermore, although we did not find an existing computer system that we felt adequately handled the investigator's data management and analysis problems while also being sufficiently flexible, economical and accessible, we were pleased to find that several systems (e.g., PROPHET, TOD, MUMPS, GEMISCH[4-8] and commercial time sharing systems) had demonstrated, by dealing with important aspects of these problems, that our preliminary goals were attainable.

We wanted to ensure that we were not being misled by our own biases, that we were not overlooking important problem areas and considerations, and that our recommendations would be generally applicable to a broad set of investigators and clinical research centers who participate in NIH's General Clinical Research Centers (GCRC) Program. We also wanted to learn more about investigators' present utilization of computers. We decided that we could best accomplish these goals by conducting a formal, in-person sample survey involving a substantial number of clinical investigators and clinical research centers. This would also provide an opportunity to gather a data base for the initial design specifications of a computer system that would assist clinical investigators in their data manipulation tasks if such a system appeared to be warranted.

The detailed results and a discussion of the survey instrument development, sample selection and survey execution are presented in Reference 1. It will suffice to point out here that the particular design and execution of a survey of user needs must be sensitive to the sociology and traditions which are inherent in the discipline under investigation.

In choosing our sample GCRCs in which to interview investigators and to collect environmental information, we selected GCRCs in larger institutions known for their productivity. Analysis of our data shows that while the 23 GCRCs surveyed represent 28 percent of the centers in the GCRC program, they represent higher percentages of the beds, annual publications, number of investigators using the GCRC, number of active projects, and annual number of research-patient days.

A total of 89 clinical investigators participated in the survey. Although this is not a substantial number, those who were interviewed appear to the authors to be representative of the more active investigators at the more active research centers.

CHARACTERISTICS OF THE INTERVIEWED CLINICAL INVESTIGATORS

Below we have characterized the typical interviewed investigator, based on median or modal responses to a series of questions.

Background and training

A typical interviewed investigator received his MD in 1959.

He started using this GCRC in 1968.

He is an endocrinologist.

He is an associate professor.

He does not have extensive training in math or statistics.

He has little or no formal or informal training in the use of computers.

Modus operandi

A typical interviewed investigator frequently develops new laboratory-analytic methodologies.

He has 5 research studies under way. This does not mean that the single investigator is working on 5 projects simultaneously, but rather that a number of studies are in different states of completion.

He has studied 28 inpatients and 13 outpatients in the past year.

He works with two senior staff, two fellows and three technicians.

He informally shares his unpublished data frequently within his institution and occasionally outside his institution.

He frequently stores his data in laboratory notebooks and on flow sheets. (Flow sheets are tabular data collection forms, generally with time as the horizontal axis and with variable or measurement names on the vertical axis. The flow sheets may be very formal and study specifics or they may be simple grids with dates and variable names entered by hand.)

Use of calculators

A typical interviewed investigator uses desk calculators and manual calculating methods. Seventy-four percent of the investigators stated that they or their staff frequently or almost always used desk calculators in the analysis of their research data, while 50 percent frequently or almost always used manual methods, including slide rules, nomograms and lookup tables in the reduction and analysis of research data.

He has good access to a programmable calculator and uses it extensively. Of the investigators interviewed, 40 percent reported that they had their own programmable calculator. Most of the investigators who had programmable calculators expressed great enthusiasm for them, were pleased with the ease with which they were used, and had grown quite dependent on them.

Use of computers

Eighty-five percent of the investigators reported that they have access to computer centers for batch processing of their research data. But of these 76 investigators, only 27 indicated that they ever used that facility. Most of the computer centers that were utilized to any extent (15 out of 25) were administratively within the medical center. Regarding interactive facilities, only 51 percent of the investigators stated that they did have access to them, and of these 44, only 19 reported that they or their staff ever used the facilities. In some cases, investigators did not know that there were facilities available to them.

We tried to determine the major reasons that investigators did not use the facilities that were, at least theoretically, available. Responses fell into the following categories: Computer Not Needed In My Research—26%, Lack Of Understanding About Computers And Their Utility—20%, Lack Of Required Assistance—14%, Have My Own Computer—10%, Administrative Problems—10%, Physical Location Of Computer Is Inconvenient—10%, Miscellaneous—5%, Cost Of Computer Time—4%. Some typical comments recorded in response to this question included the following:

- Difficulties with the logistics of getting data into the computer. The problems include specifying the data items, creating disk and tape files, preparing coding forms, transcribing data, and detecting and correcting errors.

- The investigator's methods are constantly changing so that he felt it would not be worthwhile to invest the required time in setting up an automated data handling system. This perception partially results from the inflexibility of available provisions for handling variable numbers of repeated measures and specification of limits. Also, given the investigator's background, financial support and available assistance, software development is formidable.
- Incompatible media, e.g., the computer center can't handle the paper tape that is the output medium from the investigator's scintillation counter.
- Investigator has only a small volume of data to analyze and feels the computer center represents an overkill.
- The investigator is not yet ready to use the computer.
- Funds and/or people are lacking.
- The programs that the computer center makes available are not adequate to the investigator's problem.
- Aside from hardware and software reliability problems, investigators particularly mentioned the lack of stability of computer centers, i.e., there had been changes in operating systems as well as administrative changes that had obsoleted large programming investments.
- The computer center doesn't understand the clinical investigator's needs.
- The center is several miles away.
- Long turnaround on programming; this is significant in conjunction with the need for special programs to set up data files, enter data, extract data and perform analyses. By the time the programming is completed, the investigator might have lost his train of thought, found an alternative solution, or gone on to another problem.

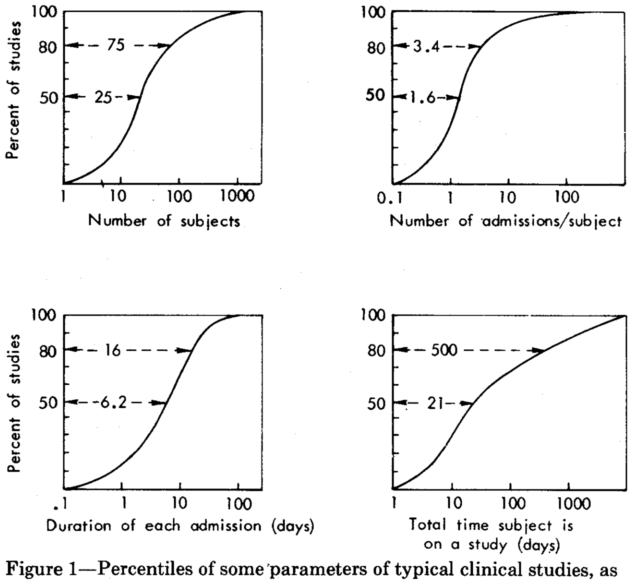


Figure 1—Percentiles of some parameters of typical clinical studies, as reported by 89 investigators

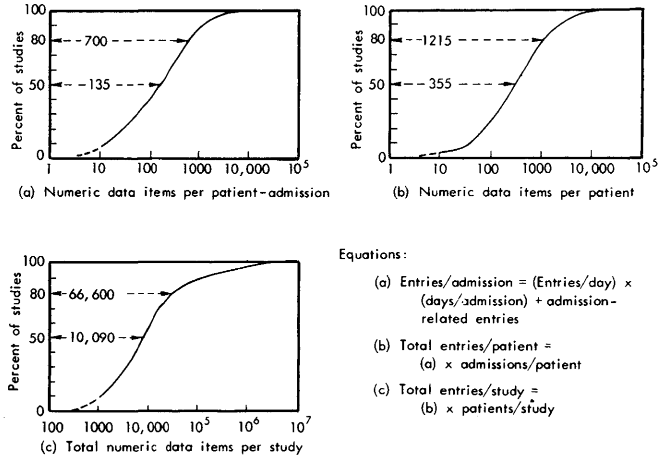


Figure 2—Percentiles of parameters of the typical clinical research data base

THE CLINICAL-RESEARCH DATA BASE

In determining the magnitude of the data base, we asked the respondent to focus on a single study that he felt was representative of his research. In most cases, the study was one in which the investigator was currently engaged.

Some of the results of these questions are shown in Figure 1. Forty-seven percent of the studies examined were inpatient, six percent outpatient, and 47 percent mixed. Many of the values in the tails of the "Number of Subjects" and "Total Time..." distributions belong to the latter two classes. However, these values have little effect on the median and 80th percentiles. To handle 80 percent of the reported studies, a computer system would have to provide (for each current study) storage for about 75 subjects per study, with three or four admissions per subject, about two weeks per admission, with each subject remaining in the study for about a year.

Another series of questions allowed us to compute the number of data items collected in a typical study. The calculated frequency distributions are shown below in Figure 2. While the 80th percentiles represent possible design criteria it is assumed that these are underestimates. Since it is not always possible to make clear distinctions between data that are of a pure research nature and data that are required for the treatment of the patient, a useful computer system would have to provide for the storage and retrieval of some nonresearch data. This would probably include patient identifying information, some medical history data, and some physical examination data. To be reasonably safe then, a system designer should think in terms of 100,000 items of numeric information (400K bytes), per study and 5,000 to 10,000 nonnumeric items per study (300K bytes).

IMPEDIMENTS TO CLINICAL RESEARCH

We and our colleagues had previously generated a model comprising 14 identifiable research processes. We

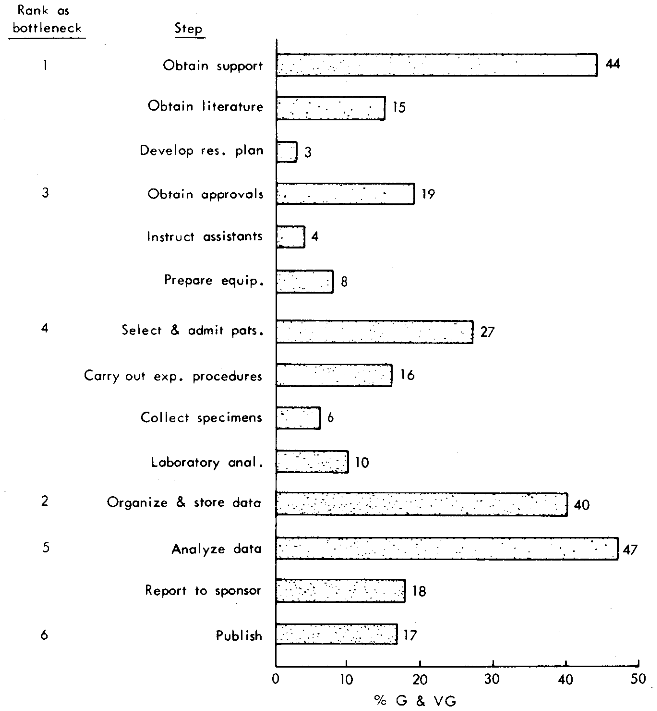


Figure 3—Percentage of GCRC investigators responding Great or Very Great (impediment) to a list of possible steps in the clinical research process

asked the respondents to indicate the degree to which each of these process or steps **unnecessarily** impeded their research. Carry Out Experimental Procedures And Care For Patients, Collect Specimens, and Carry Out Laboratory Analyses were rated low as impediments to research. This can be attributed to the fact that the studies were undertaken in GCRCs which were specifically designed, in terms of personnel and facilities, to minimize such difficulties. In our sample, we could find no convincing difference in responses to the "impediment" questions between computer users and nonusers. Retrieve, Reduce, And Analyze Data received the highest median "impediment" score. This may be related to the fact that while widely used programmable calculators may have been quite sufficient for the data analysis tasks of the investigators, they were unable to lend much assistance in the areas of data manipulation, storage, and retrieval. Figure 3, which illustrates how the interviewed investigators view their difficulties, can be used in setting priorities for exploring the problem areas in greater detail. The rankings refer to the frequencies with which the steps were named as the most severe bottleneck to research.

THE NEED FOR A DATA-ORIENTED COMPUTER SYSTEM

Several series of questions were aimed at determining the relative importance and the relative need for various features of a computer system for clinical research. The percentages of investigators who responded "frequently" or "almost always" to the question, "When you or your staff are analyzing your research data, how commonly do you perform the following procedure?" were: Graphing And Plotting—97%, Manual Transcription—96%, Descriptive Statistics—89%, Subsetting (i.e., selecting research subjects with common characteristics)—57%, Complex Statistics (e.g., regression analysis)—46%, Arithmetic Preprocessing Of Data—37%, and Modeling And Simulation—12%. It is interesting to note that the most frequently performed tasks are also very amenable to computerization.

Another series of questions asked how much of a problem it was for the investigator or his staff to perform each of a list of data manipulation tasks and operations. Figure 4 shows the percentage of investigators who responded "great" or "very great" (problem) to the indicated item. Only those items receiving scores of 25 percent or more are shown. Many investigators indicated that Develop Computer Programs represents a "great" or "very great" difficulty, but many others said it was not a problem because they never did it.

We asked those investigators who used computers about the extent to which presently automated tasks would be more difficult without the use of computers. A typical response of those investigators who used computers for tasks such as complex statistical analysis and modeling and simulation was that without computers it would not be possible for them to pursue their current research programs. Some of the tasks that computer and programmable calculator users said would be "very much more difficult" without their tools were: Complex Statistics (38 of 44 users who performed this task), Simple Descriptive Statistics (26 of 46), Subsetting (24 of 35), Sorting Research Records (20 of 28), Finding All Patients With A

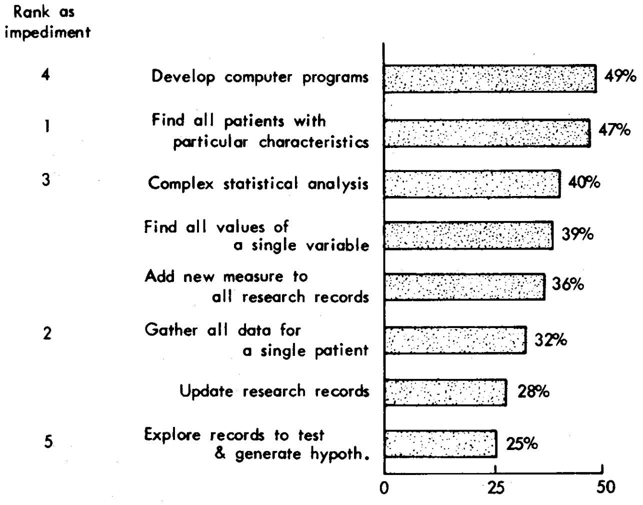


Figure 4—Percentage of GCRC investigators who rated various data manipulation tasks as being Great or Very Great problems

Particular Characteristic (19 of 28) and Modeling And Simulation (17 of 21).

BENEFITS OF COMPUTER-BASED DATA MANIPULATION

Another series of questions required that the investigator indicate the extent of his agreement with a series of statements which asserted that certain improvements (in data manipulation) could critically influence the performance of his research. A majority of investigators indicated that they agreed, or strongly agreed, with nine of the 14 assertions. The improvements which apparently can critically influence the performance of clinical research are: More Insightful Ways of Examining Data (79% agreement), Easier Retrieval Of Research Data (73%), Easier Combination Of Old And New Data (71%), Easier Subsetting Of Data (70%), Detection Of Missing Or Inaccurate Lab Determinations (65%), More Convenient Arithmetic Processing (60%), Reduced Need For Transcribing Data (57%), Easier Statistical Analysis (56%) and Quicker Notification Of Missing Samples (54%).

Three of our assertions that did **not** meet with majority agreements are: More Accurate Control Of Sample Collection Times, Less Delay In Receiving Results Of Laboratory Determinations and More Accuracy In The Recording Of Laboratory Data. Again, this appears to reflect the investigator's satisfaction with the sample collection and laboratory facilities which they have at their disposal. Also, the areas in which it was felt that improvement could have a critical effect on clinical research are those (named above) to which computer technology can make some substantial contributions.

Other questions sought to determine to what extent the investigators agreed that improvements in these data collection and manipulation areas could have certain beneficial results. The potential benefits agreed to by a majority of the respondents were: Reduced Nonproductive Time That An Investigator Must Spend On A Study (85%), Greater Insight Into The Investigator's Research Data (84%), Reduced Elapsed Time Needed To Complete A Study (75%) and Increased Publishable Results (73%). The majority of respondents disagreed with assertions that such improvements would reduce the required number of experiments, subjects or data samples. Thus, the investigators felt that the improvements would be in the form of better and more efficient use of the available clinical material rather than a reduction in the need for that material.

REQUIRED OPERATIONAL CHARACTERISTICS

Next, a series of questions asked specifically the extent to which the investigators agreed with the need for certain operational characteristics of a proposed computer system. Those features for which 50 percent or more of the respondents answered "agree" or "strongly agree" are shown in Figure 5.

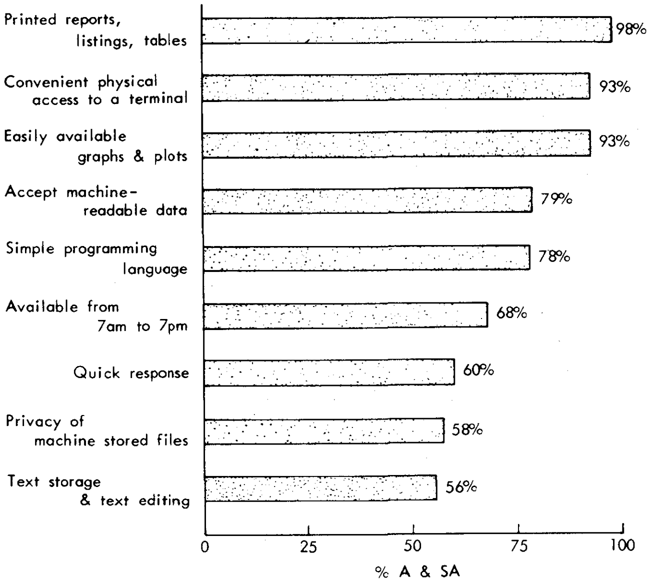


Figure 5—Percentage of respondents stating that they agreed or strongly agreed with the need for specified features of a proposed clinical research computer system

Some of the most interesting results are among the features that did not generate much enthusiasm. In particular, only 31 percent "agreed" or "strongly agreed" that visual displays of graphs and plots on a TV-like screen must be available. This is particularly significant in light of the degree to which graphing and plotting are performed as well as the expressed criticality of having more insightful ways of examining data. Our interpretation of this result is that a majority of investigators have had little experience with online graphic systems and were unfamiliar with their capabilities and advantages. This would also be consistent with the heavy stress on the need for printed reports and printed graphs and plots. Another is that investigators prefer hardcopy plots, which have traditionally been prepared by technicians or data clerks, so that they can examine and compare them at their convenience.

CONCLUSIONS

Our conclusions are primarily drawn from our analysis of the objective responses, colored by our prior informal interviews, visits to clinical research institutions, and discussions with the CLINFO clinical contractors; and by our conceptual view of the significant role that information processing plays in clinical research.

The status quo

Many investigators have and use small accessible computers and programmable calculators for arithmetic processing of their research data and for the computation of descriptive statistics. Those devices do not provide for the flexible storage or retrieval of research data files; their accessibility and their ease of use partially counterbalance these deficiencies, however.

The investigator has at least nominal access to large computer centers based administratively either at an affiliated university or at the medical school. These centers are little used by the researchers we surveyed. There appear to be three primary explanations for this. First, most investigators have had little or no training in the mathematical sciences or in the use of computers.

Second, the computer centers can rarely provide the kind of interested and knowledgeable assistance that the clinical investigator requires. Most computer center staffs either are not aware of the needs of their local clinical research communities or view them as unlikely prospective customers. Information concerning the use of the center, available programs, methods of data entry, etc., usually is not available at a level useful to the uninitiated.

Third, partially as a result of the conditions mentioned above, the investigator has little motivation to make what he views as a major investment of his career time to achieve the questionable advantages of machine-aided information processing. The barriers that he often faces include inadequate software, uncertainty about an unstable computer center environment, inconvenient geographic distance, slow turnaround (in both computer runs and special programming), and ad hoc consultative assistance lacking continuity and compatibility. In addition, he views his information processing needs as rapidly changing and requiring great flexibility. As awkward and unresponsive as his manual methods may be, he can keep personal control over his hard-won research data, can peruse it without the need for an intermediary, and does not fear its loss due to a programming error or disk crash.

As a result, the clinical investigator forgoes the potential of easy retrieval and manipulation of his study data. He also recognizes that the waste of his time and talents unnecessarily prolongs the duration of his studies and probably reduces the quantity and quality of publishable results.

Immediate need for people

The survey results show that clinical investigators could make much better use of existing computational facilities if they had better access to well trained people who are capable of educating them about the use and capabilities of computers; who are able (on a long term basis if necessary) to assist them with data organization, data reduction, retrieval, and analysis; and who can promote their use of an existing facility while providing a realistic appraisal of its capabilities and shortcomings. Also, these people must be responsive to the investigator's needs and sensitive to problems unique to clinical research.

Since few such well trained, dedicated people exist, additional support should be provided for:

- Training of laboratory and other personnel already participating in clinical research.
- Utilization of existing institutions such as biostatistics or biomathematics departments and computer centers.
- The establishment of national resources devoted to satisfying these needs.

Long-term need for computer systems

Based on our observations, substantial benefits would result from the development of a small, economical, accessible, widely-used computer system especially designed for the clinical research environment. Such a system would facilitate difficult data-oriented tasks that currently are not perceived as practical. A flexible system would encourage the sharing of data through the implementation of agreed on variable definitions, normal ranges, and data organizations while allowing the individual investigator to specify his own data entry, analytic and other procedures. Widely used systems, based on the same hardware and underlying software would encourage the development of sharable computer programs, as appropriate, while still satisfying the specific needs of particular individuals and centers. It would be extensively utilized because of its accessibility and if it were accompanied by people to assist and educate the prospective users. Also, if its cost were compatible with the research center's budgetary constraints and if it were under the center's administrative control, it would not be as subject to fiscal, administrative, and technical disruptions and uncertainties as are large computer systems operated by traditional computer centers.

While a preliminary system design may be based on the information we have so far accumulated, only experimentation in the clinical research environment will reveal the acceptability and value of a specific system design to clinical investigators. We will carry out such experimentation during the next phase of the CLINFO project.

ACKNOWLEDGMENTS

The research reported on in this paper was supported by the National Institutes of Health under Contract No. N01-RR-2-2106. This survey would not have been possible without our earlier extensive discussions and information gathering with Drs. T. G. Christopher (U. of Washington), A. W. Nunnery (U. of Oklahoma), H. K. Thompson, Jr. (Baylor), and N. Z. Shapiro and W. L. Sibley (Rand); the guidance and participation of Drs. W. G. Walker (Johns Hopkins) and R. S. Goldsmith (formerly Mayo Clinic); the support and encouragement of Drs. W. R. Baker, Jr., W. DeCesare, W. F. Raub (NIH) and B. Schachtel (formerly NIH); the other interviewers E. Fairbrother and L. Heiser (Rand) and J. Farquhar and A. MacInnes (formerly Rand); and the numerous clinical investigators and other clinical research personnel who spent their precious time with us.

REFERENCES

1. Palley, N. A. and G. F. Groner, *A Survey of Clinical Investigators and their Information Processing Activities,* R-1539-NIH, The Rand Corporation, July 1974.
2. Sibley, W. L., M. D. Hopwood, G. F. Groner and N. A. Palley, *A Prototype Data Management and Analysis System for Clinical Investigators: An Initial Functional Description,* R-1621-NIH, The Rand Corporation, August 1974.
3. Groner, G. F., M. D. Hopwood, N. A. Palley, N. Z. Shapiro and W. L. Sibley, *A Plan for the Development and Evaluation of a Data Management and Analysis System for Clinical Investigators,* R-1542-NIH, The Rand Corporation, August 1974.
4. Ransil, B. J., "Applications of the PROPHET System in Human Clinical Investigation," *AFIPS Conference Proceedings,* Vol. 43, 1974, pp. 477-483.
5. Castleman, P. A., et al., "The Implementation of the PROPHET System," *AFIPS Conference Proceedings,* Vol. 43, 1974, pp. 457-468.
6. Fries, J. F., "Time-Oriented Patient Records and a Computer Databank," *JAMA,* 12, 222, December 18, 1972, pp. 1536-1554.
7. Greenes, R. A., A. N. Pappalardo, C. W. Marble and G. O. Barnett, "A System for Clinical Data Management," *AFIPS Conference Proceedings,* Vol. 35, 1969, pp. 297-305.
8. Lloyd, S. C., B. A. Brantley, W. W. Stead and H. K. Thompson, "A Generalized Medical Information System (GEMISCH) for Practicing Physicians," *National Conference of the Association for Computing Machinery Proceedings,* August 1971, pp. 684-692.

The C.M.A. information base—A beginning of operational systems in Canada

by JAN F. BRANDEJS
The Canadian Medical Association
Ottawa, Ontario, Canada

"*. . . men who hold incommensurable viewpoints should be thought of as members of different language communities and their communicating problems be analyzed as problems of translation.*"

Thomas Kuhn:
"The Structure of Scientific Revolution"

HISTORICAL BACKGROUND

Although we can trace the beginning of the pre-paid medical care in Canada as far back as the 17th Century (Master Surgeon contract to provide medical services to several families in the community of Ville Marie—now Montreal), the nationwide acceptance of pre-paid medical care insurance plans took a relatively short period i.e., 25 years from 1946 to 1970.[1]

The first government sponsored medical plan, however, was introduced in 1962 in Saskatchewan, against strong opposition from the Saskatchewan Medical Association which eventually resulted in the withdrawal of services and a doctors' strike. Saskatchewan physicians opposed the socialization of the profession and requested the necessary amendments to secure the freedom of choice for both the patients and physicians. The Socialist government of the province was defeated, but the medical insurance plan was accepted by the newly elected Liberal government of the province because of the public appeal.

The Federal Medical Care Act of July 1, 1968, embarked the country upon a costly program of universal health care. Health care costs in Canada in the mid-1950's represented about 3 percent of the GNP and is presently rising toward 7.5 percent or more.[2]

The federal government's aim has been the "best" medical care for Canadians and has committed itself to sharing the expenditures with the provinces up to 50 percent providing the following four criteria are fulfilled by provinces:

(a) comprehensive coverage for all medically required services;
(b) universal availability to all eligible residents of the province; there had to be at least 90 percent of the eligible population covered;
(c) portability of benefits between provinces and elsewhere (in the United States and Europe);
(d) and it had to be publicly administered by the provincial government and non-profitable.

At the present, all ten provincial governments have only a single paying agency, (commission or board) using well-organized data processing for assessment of claims and billing purposes, and as a comprehensive data base for provincial, federal statistical and research departments. Both physicians and provincial medical associations obtain reports comprised of details of every medical service submitted for assessment and payment.

For many physicians, computerization and control represents "socialization" of medicine; for others, a threat to free market principles, despite the prevalence of the fee-for-service remuneration.[3] In order to minimize the confrontation between government paying agencies and organized medicine during negotiations for fee schedules, working conditions and such, the Canadian Medical Association (C.M.A.) has been establishing a computerized information base, which will contain both statistical and narrative information relevant to medical care.

The concept, development, structure, maintenance and utilization of the C.M.A. information base has originated from the early works of information scientists such as Luhn[4] and Borko[5] and is based on a free-text searching technique.

A BRIEF ANALYSIS OF INFORMATION SYSTEMS

Information systems, from the classical point of view of an information scientist, may broadly be structured into two levels: information storage and retrieval systems (ISR), and control and management information systems (MIS). Both types of systems are well-covered in terms of published materials. Surveys of bibliography on information storage and retrieval systems may list well over 5,000 items, and a recent survey of bibliography on management information systems consists of no less than 2,000 items.[6]

In the past, information scientists, systems and management specialists were deluded by computers, by the intricacy of programming and by the refinement of interesting but not too practical mathematical and analytical methods. Therefore analyses of information flows within individual parts of systems resulted in total misunderstanding of the drastic changes under way.[7] More important, the major goal of any information system, a satisfied user, has almost been omitted from the research, analysis, design, implementation, retraining and adjusting processes. For example, in the design of "total" hospital information systems (HIS), all aspects of hospital administration were included except the patient and his needs.

Does this mean that viable information systems (both ISR and MIS) are a mirage which will never be achieved? Bishoff[8] in his work "Die Informationlawine" points out that there is an urgent need for effective information storage and retrieval systems due to the fact that information in some areas of human knowledge doubles at the following rates:

General information	—every 10 years
Information on chemistry	—every 8 years
Information on electronics	—every 5 years
Information on space	—every 3 years

The well-known "publish or perish" syndrome—a very important factor in the lives of all academics and researchers as a means for promotion and fame, has contributed significantly to the abundance of information by the publication of many articles, papers and even books dealing with topics which have no relevance to the real life information. Dr. Shires, Dalhousie University, observes, "What is of greater concern is not the explosion (of information) but data pollution so that the information content becomes strangled."[9]

Information scientists are aware of the emergence of the information explosion (or pollution) since Luhn's permutated indexing techniques developed in the early 1960's. They tried to mechanize and later to automate libraries by means of better housekeeping which included the cataloguing, indexing and autoindexing, filing, searching, ordering and information dissemination. Many systems for university libraries and other knowledge-based institutions have been launched, all but few seem to fall far below initial expectations. Only very recently (and after a decade of efforts), MEDLARS, a typical batch processing medical library system, offered an online version of bibliographical search called MEDLINE. The general trend of library scientists has been to cope with mainly the methodological, and procedural aspects of the library rather than with the needs of the users.

The Canadian Medical Association, Department of Research and Development has been building a comprehensive information base over the past two years in order to supply relevant information to its divisions, scientific councils and staff. The conceptual development of the C.M.A. information base has been designed on the following, somewhat paradoxical, analytical findings of the state-of-the art:

1. Despite tremendous efforts in terms of money and thoughts, there are very few successful computerized information systems available to the user;
2. In the rapidly increasing volume of printed materials, books, monographs, textbooks, journals, papers, etc., it appears that the content of knowledge is decreasing because of the lack of adequate quality measures;
3. High expectations have been placed in computers, mathematical, linguistic and information applications, all but few are of very little use in terms of real world projects;
4. It seems that the major reason for the past failures of information systems stems from the misunderstanding of the end-user's needs in the current systems era.[10] Until now all efforts were dedicated to the technology in the broadest sense, i.e., computers, communication, programming and terminals;
5. Despite all the failures, there are a few successful applications which maintain the necessary level of optimism and promise for the future (e.g., ATS/370, QL System, JURIS, MARK IV etc.);
6. Both researchers and practitioners have to accept the fact that traditional printed documents are, and will be for a long time to come, the prime source of information.

In short, to avoid the past failures of many computerized information systems, any viable approach to information processing, namely to both input and retrieval, must be human-oriented. It should follow the path of the learning processes, methods of research, methods of annotations, techniques of writing and similar intellectual activities.

MANAGEMENT OF INPUT

An ideal transformation of information into the computer would be a direct input from the material prepared for publishing by means of a computer-aided typesetting or, in our case, from the tapes of government computers. The computer-to-computer input technique is still in the stage of new developments and therefore is, unfortunately, not yet available. The next best method of input to follow is that in which most scientists, technical writers, and researchers have built their private, personalized information storage and retrieval files aiding memory by notes, manual files, cards and so forth. When studying articles and monographs, or obtaining any information in the sphere of a specialist's interest, the source of the information, name of the work, index (keyword) and the content is registered in a more personalized way than the usual bibliographic annotation. An experienced user of information also tries to note the "flavor" of the content, namely the richness of the author's language, new ideas presented, contribution to knowledge or personal enrichment. Each user, however, may have different styles of building up the necessary knowledge, as well as, different approaches to this process of information storage. Therefore some sort of common classification is necessary.

For example, the classification of inputs in the C.M.A. information base should meet the present needs and future requirements of researchers, medical politicians, mainly practicing physicians, provincial divisions and C.M.A. staff. It eliminates, to a bare minimum, digital codes and uses free English text in most cases. (Exhibit 1) A decimal coding system composed of classes "0" through "9" seems to be a good choice fitting most C.M.A. research and information needs. Thus, group "0" is assigned to cover general knowledge, while group "9" represents a very specialized field of interest. An example of group classification is shown in Exhibit 1. This classification states boundaries of interest and the extent of the C.M.A. information base and acts as a common denominator for all users. To specify further users' needs and attach some quality yardstick to information stored, a sub-classification of each group is required, as follows:

Sub-Class	*Content*
1	Phrases and expressions of elementary quality: good sentence structure, straightforward thinking of experienced practitioners in the field.
2	Phrases and expressions of intermediate quality: richer language, expressions not so often seen.
3	Advanced use of the language: fresh ideas expressed in new ways, perhaps not yet clearly understood by the reader and set aside for the second reading.
4	Citations and quotations.
5	Jokes and statements for warming up presentation.
6	Formulas, algorithms and simple graphs.
7	Criteria and tables.
8	Headlines of interesting topics and ideas.
9	Definitions, thoughts and glossary of terms.
0	Bibliography and references.

It should also be stressed, however, that these numeric codes are for the classification of stored information only and are not required in the query and information retrieval processes.

The input coding form used is just a little more structured form of usual manual notes, excerpts and indexes. However, any kind of notes may be used if understood by the person entering the information. There are no rules or requirements for the input other than the identification of the group and subclass.

Each input document consists of four entities:

1. Group (Class) Information Number, e.g. 00-general knowledge or 60-economics
2. Subclass Number, e.g. 9-ideas or 4-citations, and such
3. Text of Document
4. End-of-Document signal character

ACCESS TO THE C.M.A. INFORMATION BASE

After proper identification through one of three types of terminals tested—two video (an IBM 3275 CRT and Bell Vucom I) or via a typewriter terminal (IBM 2741)—the users of the C.M.A. information base may access the computerized information base. The search query may be entered in narrative form. For example:

```
I WANT INFORMATION ON THE IMPACT
OF INFLATION ON MEDICAL PROFESSION
IN BRITISH COLUMBIA
```

The system analyzes the statement for what it considers significant terms and searches the information base for all document references containing all or any of these terms. The common words such as I, WANT, ON, THE have been "stopped". This means they are regarded as non-significant and the system will not search on them.

It is preferable, however, to enter the search query as a string of significant words.

```
IMPACT INFLATION MEDICAL PROFESSION
        "BRITISH COLUMBIA"
```

A message to the user,

```
YOUR SEARCH IS PROCEEDING
```

is displayed. Within 3 to 30 seconds the information is displayed. If the search query contains, in addition to valid search terms, words that do not occur in the data base the message

```
THE FOLLOWING WORDS IN YOUR QUERY DO
NOT OCCUR IN THIS DATA BASE:
              BRITISH COLUMBIA
TO RESTATE YOUR QUERY, TYPE "S". TO
CONTINUE, PRESS "ENTER".
```

is displayed. This gives the user the option of proceeding with the search or returning to the search mode to restate the search query.

If the search query contains only words that have been stopped or words that do not occur in the information base the message

```
THE WORDS YOU ENTERED ARE EITHER TOO
COMMON OR ARE NOT FOUND IN THIS DATA
BASE. PLEASE RESTATE.
```

tells the user to restate the search query.

The user may consult the dictionary to see what words occur in the information base and whether they have been stopped.

Search terms can be combined using "OR", "AND" and "BUT NOT" logic. "OR" is represented by one (or more)

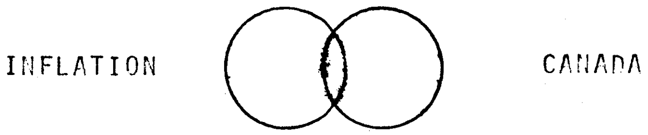


Figure 1

space(s). The command

INFLATION CANADA

means the user wants references to documents in which the words 'inflation' or 'Canada' occur. Figure 1 illustrates the "OR" relationship.

"AND" is represented by & (ampersand). The command

INFLATION & CANADA

means the user wants information on both 'inflation' and 'Canada'. Figure 2 illustrates the "AND" relationship.

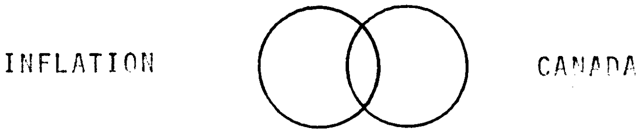


Figure 2

"NOT" is represented by % (percent sign). The command

INFLATION % EUROPE

means the user wants references to documents about inflation but wishes to exclude those in which the word 'Europe' also occurs. Figure 3 illustrates the "BUT NOT"

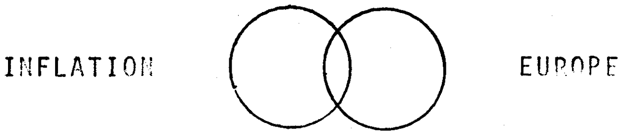


Figure 3

relationship. Similarly, a mandatory "AND" (&) can be used to link groups of words. If the search is phrased

MALPRACTICE & MEDICAL DOCTOR HOSPITAL

the system will retrieve documents in which the word "malpractice" occurs and in which one or more of the three words "medical', 'doctor' or 'hospital' occur. The "&" merely requires that one of the terms on either side of the "&" symbol occur in the document. If the search is phrased

MEDICAL & MALPRACTICE DOCTOR HOSPITAL

the system will correctly retrieve information on 'medical' and 'malpractice', but may also retrieve documents in which 'medical' and 'doctor' occur or in which 'medical' and 'hospital' occur.

QL PROGRAMING SYSTEMS

The C.M.A. information base management is using the Quick Law (QL) Systems. QL Systems Limited retrieval system operates on IBM System/360 or System/370 configurations under the IBM Operating Systems OS/MFT, OS/MVT, OS/VS1 and OS/VS2. The retrieval system consists of several program modules; all of these modules are written in System/360 Assembler Language. The File Build programs are also written in System/360 Assembler Language, except for the program which sorts the concordance. For this latter program, the OS Sort/Merge utility program is used. The 'driver' program, which is used to create the input to the File Build programs, can be written in any System/360 programming language. Most of these 'driver' programs are currently written on either COBOL, PL/1 or Assembler.

There are three File Build programs. The first program generates the Text data set (the document itself), the Text Index file (a pointer to the document in the Text data set) and the concordance (a list of all words which appear in the information base). The second program sorts the concordance (produced by the first program) and creates the Dictionary data set and the Dictionary Index data set. The third program generates the Locator data set (each entry in this data set identifies a document section in which a particular term occurred, how many times the term appeared in this document section, and algorithm values which indicate the statistical significance of the term in the information base).

Exhibit 2 illustrates the sequence of events involved in the creation of an information base.

TEXT DATA SET

The Text data set is made up of the text of the document itself and directory entries. There is one 8-byte directory entry for each section of a document and documents can be subdivided into 8 sections. The directory entry for a section contains the following information: the directory block number, the length of the text section, the type of document section and the number of lines in the document section. These directory entries are appended to the end of each document in the data base.

Within the text of the document itself, blanks (or spaces) are placed by a one-byte 'hash code'. This hash code varies according to the word it precedes. The hash code is used for both text searching and for automatic highlighting of terms. The use of a hash code before each word in the text (i) removes the need to store each possible phrase which occurs in the information base and its associated pointers (e.g., pointers to document sections) and (ii) removes the need to compare each entire word or phrase in the document. Instead, the retrieval system will look for hash codes which match the hash code associated with the terms of the search. This facility decreases the amount of time required for a search and also decreases the amount of direct access storage required.

Text index data set

The Text Index data set consists of one 4-character pointer (to the start of the document directory) for each document in the information base.

Dictionary data set

The Dictionary data set consists of dictionary entries and there is one entry for each term or word in the information base. The entry contains the following information: the word (up to 16 characters), the number of times the word occurs in the information base, the number of documents which contain the term, a pointer to the locator data set, and a pointer (to another dictionary entry) which is used for synonyms. Each dictionary entry is 28 characters long.

Dictionary index data set

The Dictionary Index data set contains pointers to the Dictionary entries. The presence of his file removes the need to search the dictionary entries for a particular search term.

Locator data set

The entries in the Locator data set are 12-character long and contain the following information: the number of times the term occurs in a section of a document, the number of the documents in which it occurs and 5 values reflecting the significance of the term in the information base depending on the method or algorithm used to compute these values.

The values are computed only once (when the information base is built) so there is no need to repeat these computations whenever a particular term is used in a search. There is one Locator entry for each document section in which the term occurred. No Locator entries appear for words which are common in the information base.

TECHNOLOGICAL SYSTEMS

Machine requirements

The following configuration is the minimum machine requirement for the assembly and/or execution of the retrieval system programs:

- one CPU IBM System/360 Model 40, or System/370 Model 135, with core capacities of 50 K in addition to the core requirement for the Operating System. On IBM 360 series computers, core may be Large Capacity Storage core rather than main core.
- one console
- one card reader and printer
- one IBM 2314 or 3330 Disk Storage Facility (or equivalent device)
- one IBM 3275 or IBM 2741 or teletype compatible terminal
- one 2701, 2702, 2073, 3704 or 3705 communications controller

Size of C.M.A. information base

There are no rules for computing the space requirements and costs for a particular section of the C.M.A. information base. However, the following interrelated factors are significant:

- the size of the text
- the number of different terms which appear in the information base
- the number of occurrences of the terms in the information base
- the number of documents in the information base

Generally speaking, the size and costs of the information base will vary directly with (i) the size of the text (ii) the number of different terms which appear in the information base (iii) the number of occurrences of the terms in the information base and (iv) the number of documents in the information base.

Once a certain size has been reached, neither the Dictionary or the Dictionary Index data sets will expand. Obviously, as documents are added, the Text and Text Index data sets will increase in size. The Locator data set will also expand, but much less rapidly that the size of either the Text or Text Index data sets.

For a very large information base (i.e., an information base which has a large amount of text and many documents), the amount of storage overhead (the space occupied by the Text Index, Dictionary, Dictionary Index and Locator data sets) may be less than or equal to the amount of storage occupied by the Text data set. For a very small information base (i.e., an information base which has a small amount of text and few documents), the amount of storage overhead may be a factor of three or four times the amount of storage occupied by the Text data set.

CONCLUSION

This on-going research project of the Canadian Medical Association represents the advent of interactive, online computer-centered information sharing; a technological advance that is expected to change, and hopefully revolutionize mental attitudes of researchers, medical practitioners, health care politicians, administrators and students of systems and health care sciences.

Online computer-based information systems have only recently achieved sufficient technological maturity. It is the role of the end-user to implement new forms of creative information processing superior to the traditional batch data processing.

REFERENCES

1. Government of Canada, *The Federal Medical Care Program*, National Health and Welfare, Health Insurance Directorate, Health Program Branch, Ottawa, 1974.
2. Report of the Canadian Computer Task Force, *Branching Out*, Vol. 2, Ottawa, 1972.
3. Brandejs, J. F., *The Health Care Plans of Canada*, paper, The C.M.A., 107th Annual Meeting, Toronto, 1974.
4. Magnino, J. J., *IBM's Unique but Operational International Industrial Textual Documentation System (ITIRC)*, International Congress on Documentation, Tokyo, 1967.
5. Borko, H., Evaluation of the Effectiveness of Information Retrieval Systems, *Proceedings of the IFIP Congress*, 1962.
6. Cuadra, C. A. (editor); *Annual Review of Information Science and Technology*, Vol. 1-7, (1966-1973), American Society for Information Sciences, Encyclopaedia Britannica, Chicago, Illinois.
7. Ackoff, R. L., "Towards a Behavioral Theory of Communication," *Management Science*, Vol. 4, No. 218, 1958.
8. Bishoff, J., Die Informationslawine, Oldenbourgh, Dusseldorf Wien, 1967.
9. Shires, D. B., *Computer Technology in the Health Sciences*, C. C. Thomas, Publisher, Springfield, Illinois, 1974.
10. Brandejs, J. F., *Health Informatics*, internal publication, The Canadian Medical Association, Ottawa, 1974.

EXHIBIT 1—SECTIONS OF THE C.M.A. INFORMATION BASE

Each section may have a different format of documents entered because C.M.A. information base is user-oriented and not library-oriented with a fixed structure of documents stored.

For example, in 'CMA GEN', all kinds of information might be stored in free text form. The search will be oriented toward frequency of words searched for. On the other hand, in 'CMA SPE' will be only statistical tables, and only the headings are searched for.

Documents need to be labeled and listed. For example, 'GEN D1' will be a list of documents entered in the general section. Search will therefore be done in the following way:

Sign on: 'CMA GEN.' Search for any information, then ask 'D1' which will give list and names of documents in the file.

Information base sections:

(a) GEN—general file containing:
1. documents explaining information base structure and how to search information base,
2. all documents which are general and applicable to all Councils and accessible to anyone, for example:
   (i) all topics in "The Way I See It"
   (ii) list of all conferences
   (iii) list of meetings in divisions and of affiliate societies

(b) SPE—specialized data file containing:
1. raw data—of any statistical significance
2. data, compiled in tabular form
3. tables of trends
4. results of surveys classified as follows:

SPE 0	National tables
SPE 1	British Columbia
2	Alberta
3	Saskatchewan
4	Manitoba
5	Ontario
6	Quebec
7	New Brunswick
8	Nova Scotia
9	Prince Edward Island
10	Newfoundland
11	Northwest Territories and Yukon
20	United States
30	Europe
90	World

(c) CON—information file on—scientific councils containing:

CON 0	Transactions—present index of policy statements
CON 1	Board of Directors
CON 2	Council on Community Health
CON 3	Council on Medical Services
CON 4	Council on Medical Education
CON 5	Council on Medical Economics
CON 6	Membership Services
CON 7-8	Vacant
CON 9	R & D

(d) EXL—exclusive information base only for C.M.A. use (blocked to outsiders)

EXL 9—KIKO: each document is divided into four segments and each can be retrieved independently:

segment 1 subclass (1 digit)
segment 2 groups (2 digits)
segment 3 KWOC or KWIC
segment 4 text

Detailed structure of EXL—KIKO

Group 0 GENERAL AND BASIC KNOWLEDGE ABOUT MIS

(all information related to computer-aided systems)

00	General Management Sciences (concept of scientific management)
01	Computer-based MIS
02	Hardware (computers)
03	Software, including Data Base

04 Information Systems for the Management of Computer Centers
05 Computer-aided Medical and Health Care Information Systems
06 Soft-copy oriented (VMT) Information Systems
07 Model base, Methodological Base, and Information Retrieval Systems
08 Education of Computer-based MIS Users
09 Behavioral Aspects and Psychology of MIS Users.

Group 1 INDUSTRIAL ENGINEERING
10 General Knowledge of IE
11-18 Vacant
19 MIS Engineering

Group 2 MIS FOR PRODUCTION
20 Production-oriented MIS: general knowledge
21-29 Vacant

Group 3 MISCELLANEOUS
30 Bylaws, acts, bills
31 Meetings, manuals, directories, handbooks, tables
32 Journals, reports, bulletins, records and pamphlets

Group 4 VACANT

Group 5 HEALTH CARE
50 General ideas
51 Management of Health Care and Nursing Care
52 Medical Care (Biology)
53 Lifestyle Health Care

54 Environmental Health
55 Health Policies; MEDICARE, Health services, Insurance, Group Practice
56 Health and Medical Statistics
57 Physician Information—(fees, distribution, classification)
58 Vacant
59 Privacy and confidentiality

Group 6 ECONOMICS
60 Economics in General and Budgets
61 Medical and Health Economics and Expenditures
62 Manpower, planning & needs
63 National income & expenditures, and financial information
64 Wages, prices and price index
65 Annual reports, briefs
66 General statistics
67 Remuneration: fee setting, gross and net earnings, overhead
68 Collective representation and Negotiation
69 GNP and Trends

Group 7 DECISION-MAKING PROCESSES
70 Mathematical-oriented D-M
71-78 Vacant
79 Theory of Policy Science

Group 8 EDUCATION
81 Education in General
82-88 Vacant
89 Recurrent Issues in Higher Education

Group 9 KIKO (Knowledge In-Knowledge Out)
90 Rules and Phrases
91 General Phrases for good writing
92-94 Vacant
95 Who is Who?
96-98 Vacant
99 Pure Medical Terms

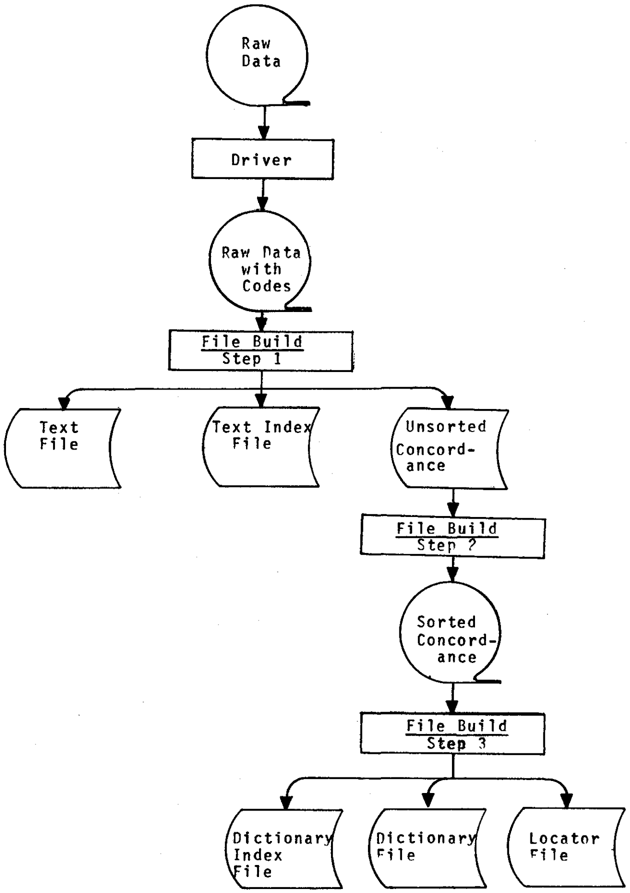


EXHIBIT 2—CREATION OF A DATA BASE

A comparative evaluation of automated medical history systems

by EPHRAIM R. McLEAN with the assistance of STEFANIE V. FOOTE
University of California
Los Angeles, California

INTRODUCTION

As the practice of medicine shifts from crisis intervention to the prevention of disease and the maintenance of health, the role of the comprehensive patient medical history becomes even more important than it has been formerly.

With the need to establish a "data base" of information on the patient's general condition, the attending physician must spend anywhere from a few minutes to over an hour asking a variety of questions and recording the responses, either in long hand or by dictation (which must then be transcribed by a medical secretary).

These questions range from those pertaining to family history (e.g., father died of a heart condition) and social habits (e.g., patient smokes two packs of cigarettes a day) to the patient's own medical background (e.g., history of jaundice) and current review of systems (e.g., the cardiovascular and musculo-skeletal systems). Also included are past hospitalizations and operations, medications the patient is currently taking, and some indication of why the patient is seeking medical attention in the first place (i.e., the chief complaint or problem).

It should be noted that this history is but the first step in the fact finding and diagnostic process. In addition, the particular problem which the patient has must be explored in depth and a physical examination conducted. Based upon these findings, laboratory tests and radiological exams may be indicated. After this, there may be more questions and more tests until the physician finally arrives at his conclusions as to the nature of the ailment and the therapy and medications necessary to treat the condition. Thus the collection of the history data is more of a prelude to the actual diagnostic process than an integral part of it. This is important, for while the physician has a central role in the decision on the diagnosis, he does not necessarily have to collect all of the information on which this diagnosis is based. Indeed, there are many examples of the delegation of these tasks to residents, nurses, and lab technicians. This "division of labor" allows each member of the medical team to specialize in that phase of patient care for which he or she was trained. It also helps define those areas in which technology can be advantageously applied.

Although the medical history is but the first step in the delivery of health care, it is a vital first step. It, in conjunction with the physical examination, provides a baseline from which to begin care and with which to gauge changes over time. This "work-up," as the patient interview and examination is called, is designed to give a complete picture of the patient's medical status, not just the problem of current concern. Unfortunately, because of the pressures of time and the desire on the part of the physician to get to the problem at hand, there is sometimes a tendency to focus exclusively on the particular problem and not ask all the routine history questions. The physician's busy schedule usually does not allow him the luxury of exploring every aspect of the patient's physical and mental condition. In every patient interview, a number of questions might be asked that would provide a more complete history, but there is simply not enough time for them. Fortunately, by training and by instinct, the experienced clinician is able to move quickly to the important facts of the patient's condition and rarely does any harm result from the few items of data that are missed. However, from the patient's standpoint, this inability of the physician to listen to all minor complaints and problems can be disconcerting.

Recently, the medical history has received attention from another quarter, that of appraisal of care. This increased scrutiny comes from two sources. First, the governmental agencies and insurance companies which function as third-party payers are concerned about the nature of the care delivered as a basis for payment. Second, the utilization review committees and Professional Standards Review Organizations (PSRO) are concerned with the appropriateness and quality of care. Both of those groups are using the medical record—of which the history is a part—as one of the prime factors in their evaluations. Thus there is increased pressure for more complete records and documentation. However, at the same time, physicians are being pressed to expand care to underserved populations and to improve care to those already being served. In the face of these competing demands, it is understandable why the assistance of the computer and information processing technology is being sought.

AUTOMATED MEDICAL HISTORIES

Although the eliciting of medical history data is most usually done by the physician, there are many instances where this task is performed by residents, nurses, or other medical assistants. These individuals question the patient, record the responses, and present the completed history to the physician for his review. A logical extension of this is for the patient to record his own history directly, thus eliminating the need for the third party. These self-administered medical histories have been in existence since World War II, first with manual, pencil-and-paper questionnaires, and more recently with the aid of computer processing. In this way the patient plays a more active part in the creation of his medical record, a role which most patients are more than willing to play. As Dr. Lawrence Weed has pointed out,[1] the patient's time and active involvement is one of the most underutilized resources in medicine.

As with any emerging technique, there is a wide range of implementations that have been tried. Some attempt to provide a complete history for the physician, going so far as to suggest possible problems (i.e., quasi-diagnoses) and to recommend lab tests, while others perform only a basic screening or triage function or focus on a particular medical area (e.g., cardiology problems).

There is a similarly large variation in the techniques of administration. The first medical history questionnaires were merely checklists which were filled out by the patient and then given directly to the doctor for him to scan. Later, some of these questionnaires were transcribed with word processing equipment like the IBM Magnetic Tape Selectric Typewriter (MTST) or keypunched for computer processing. The use of prepunched cards and mark sense forms have also been employed. With the advent of on-line systems, the possibilities for an interactive, conversational dialogue between the computer-based history system and the patient have become realized. With these, multiple levels of branching are possible, detailed explanations can be displayed when the question is unclear, and even foreign-language versions are easily implemented. In the next section, these various techniques will be described and evaluated in more detail.

TECHNIQUES OF ADMINISTRATION

In their comprehensive monograph on the *Acquisition of the History Database,*[2] Yarnall and Wakefield identified 15 different design approaches and gave illustrations of 75 different systems. Drawing upon their study and the published results of other efforts, including those of the author,* the following is a summary of the various techniques of administration. For simplicity, they can be grouped into 4 major categories: (1) manual systems, (2) non-computer machine processed, (3) off-line computer processed, and (4) on-line computer processed.

Manual systems

The first medical history questionnaire to come into general use was the Cornell Medical Index (C.M.I.). Devised by Brodman[6] in the late 1940's, it consists of a form containing 195 questions which is given to the patient immediately before the office visit. Further processing is not required, although an effort was made at one point to introduce a computer-processed version.[7] The examining physician quickly scans the patient's responses and then proceeds with his own questioning. Since its introduction, the C.M.I. has undergone almost no changes in either its composition or in question wording. It continues to enjoy widespread popularity and it is estimated that more than 300,000 are administered annually.[8]

Another widely used questionnaire is the one of the Department of Defense, used for screening recruits and in conjunction with periodic physical examinations for members of the armed forces. Known merely as DD Form 88 and DD Form 89, these two documents are similar in concept to the C.M.I.; and although restricted in usage to the military, they have been used extensively in this context for over twenty years.

The chief problem with manual questionnaires is their limited ability to provide for branching as a function of the results of previous questions. If there is more than one level of branching, the instructions can become quite complex and confusing. This means that particular problems cannot be explored in any great detail. Also, the physician must scan through the entire questionnaire, for there is no summarization for him. An offsetting advantage, of course, is that these manual versions are quite inexpensive, costing less than a dollar apiece.

Non-computer machine processed

In an effort to provide a neat typed summary, which is missing from the preceding, attempts have been made by Kanner[9] and others to use word processing equipment like the IBM MTST. Inputting can be done directly from a questionnaire or in conjunction with special filmstrip equipment which is linked to an MTST. In this latter case, there are function keys which the patient uses to record his responses.

In both of these approaches, the assistance or interven-

* These descriptions are drawn from a number of sources, including two studies conducted by the author at the Lahey Clinic in Boston Massachusetts.[3,4] For a complete list of references on the subject of techniques for administering medical histories, the author and co-author have prepared a bibliography of some 410 items,[5] copies of which can be had by writing to him at the Graduate School of Management, UCLA.

tion of a nurse or secretary is needed. Also, the cost of the equipment must be considered; but it is not uncommon for many physicians, even in solo practice, to have word processors of some sort; and thus the cost can be shared with other applications.

Off-line computer processed

A logical extension of the manual questionnaires was the move to the keypunching of the responses and their subsequent batch processing. This introduces the requirement for keypunch equipment and operators as well as the need for computer processing. For most physicians this means sending the work to a service bureau, with the corresponding problems of long turnaround times, costs, and possible loss of confidentiality. And, as with all off-line modes, there is the potential source of error due to transcription mistakes. Even if the clinic or hospital is large enough to have its own computer equipment, as is true of the Lahey Clinic in Boston, there are still problems of cost (anywhere from $3.00 to over $10.00 per patient) and the inability to handle high volumes with reasonable turnaround times.

One way to resolve the keypunch problem is to move to mark sense documents. This requires specially-prepared forms (which are fairly expensive to buy and difficult to modify) and special optical scanning equipment. However, the need to transcribe is eliminated and the Lahey Clinic, which has gone to this approach,[4] has found the error rate to be quite low while at the same time achieving same-day turnaround (less than 5% of the forms require any manual intervention to insure successful processing). At the Mayo Clinic, Mayne[10] reports similar success with the use of mark sense forms.

In one of the major efforts in utilizing the computer in support of multiphasic health screening, Dr. Morris Collen at the Kaiser Permanente Medical Center in Oakland has made use of medical history questionnaires for more than twenty years.[11] In its present form, it consists of two parts: a deck of 204 prepunched cards designed for review of systems information with a single yes-or-no question printed on each card, and a pencil-and-paper questionnaire for past history. For the first part, the patient indicates his response by dropping each card into the "yes" or "no" section of a divided letter box. The positive responses are then sorted and listed immediately for the physician's review. The responses on the questionnaire form are keypunched and are added to the patient's medical record later.

A final development is the use of terminal-like devices which capture the patient's responses on magnetic cassette recorders. These units are linked to filmstrip or carrousel projectors or even to audio devices for the presenting of the questions. The resulting cassette can then be either mailed in or telephoned in for processing, with the results mailed back or printed in the doctor's office at a remote terminal. The turnaround times in the latter case can be almost equivalent to an on-line system, but at a much lower cost.

On-line computer processed

With the growth of multiprogramming and time sharing, it now becomes feasible to consider putting the patient "on-line." In a pioneering effort at the University of Wisconsin, Slack and his coworkers[12] developed the first online computer-based medical history system. Using a dedicated LINC laboratory computer, questions were presented to patients by means of a CRT display. At the end of the session, the results were summarized and printed out for the examining physician. The use of an online mode provided the ability to have extensive branching. The response to one question would determine, to a limited extent, the next question that was to be asked.

The use of multiple CRT devices, each under central computer control, is probably the most powerful of all approaches for collecting patient history data; but it is not without its problems. The keyboards are not typically designed for patient use, the resolution of the image on the screen may leave something to be desired, costs are high, and reliability may be a problem. As Yarnall has suggested, "Expect headaches! (That's what your grant is for!)."[2]

At the Massachusetts General Hospital, Grossman, Barnett, and others[13] explored the use of Teletype terminals in an interactive mode. As with Slack's work, this system also allowed extensive branching. However, because of the limited availability of terminals, the system is still on an experimental basis.

Some of the work in off-line data collection is being carried over into an on-line environment. The use of filmstrip projectors and back-projection carrousels (using either regular slides or microfilm) are now being operated under computer control. These displays offer better resolution than those of CRT's and extensive branching is still possible. With the use of minicomputers or "intelligent" terminals, this approach may prove to be the most desirable in the long run.

EVALUATION OF AUTOMATED HISTORIES

Recognizing these wide variations in types of histories, it is difficult to compile a single list of advantages and disadvantages that would apply to all approaches. What would be an advantage from one standpoint (e.g., low cost) might be a disadvantage from another (e.g., insufficient detail). However, to fail to make any attempt at all would be to do a disservice; and so the following list must be read with the foregoing in mind.

Advantages

Time savings

The argument given most frequently in support of self-administered histories is that they save time. If the physician takes the patient's history himself, it is the doctor's time that is being saved. If the doctor dictates the history and thereby saves the time needed to write it out himself, there is still the time of the secretary who must transcribe the dictation. Depending upon the rate at which the doctor accounts for his time, the saving of even a few minutes could be worth several dollars. This time saving can be used to enable the doctor to see more patients or to use the extra time to delve more deeply into each patient's particular problem or problems.

In this latter vein, there are a number of physicians who report that the patient-prepared histories give them a "head start" on their own questioning. They feel that they can go directly tc the patient's chief complaint because most of the routine questions have already been covered.

Completeness

Closely linked with the preceding is the ability of many of the automated medical history systems to give a more complete picture of the patient than is true of those generated by traditional means. This is particularly true where the doctor is pressed for time and can only concentrate on the main problem the patient has. Even in those cases where the doctor duplicates the previously-administered medical history and reasks all the questions, there is still a feeling that the patient-produced version provides a check on the doctor's own questioning. More substantively, there are two areas where this possibility of greater completeness can be of distinct importance. The first has to do with the bringing to light of conditions which are medically important but completely unrelated to the patient's main problem. In such cases, it is possible that the secondary conditions might be overlooked in the process of arriving at the primary diagnosis.

The second is closely related to the first. With the growth of the many medical specialities, physicians are, by choice and by training, focusing more and more upon particular problems and not on the whole patient. Thus a comprehensive medical history can serve to point out other problems the patient may have which may be far removed from the doctor's own specialty. This information can lead to the patient being referred to another physician or being rescheduled for another visit.

Legibility

The many jokes which are made about physicians' handwriting might be funny if the subject were not such a serious one. No longer is it a case of the doctor who makes the entries in the record being the only one who has to read them. With the increasing need for a variety of physicians and health care providers to have access to the medical record, it becomes essential that it be readable. Here the computer-produced history provides a clear benefit over the handwritten version.

Patient participation

A concern which has been raised in some quarters is that although physicians may like the assistance that computer-aided medical history systems afford them, patients may not. However, studies indicate that the reverse is true.[14] Some patients have even expressed a preference for the self-administered histories. There are a number of reasons for this attitude.

With the pencil-and-paper versions of the history questionnaires, especially those filled out at home, patients like the less hurried atmosphere and the ability to take their time and answer each question carefully. Also, family medical records can be consulted and the labels of current medications can be checked for drug names, dosage levels, and so forth. Because most patients have come to expect to be able to spend only a brief amount of time with their doctors, the carefully-completed history reassures them that the doctor will have a complete picture of all of their problems.

For the terminal-based versions, the ability to branch forwards and backwards, to have questions explained when they are unclear, and to have a complex machine patiently waiting for each response is a stimulating and exciting experience.

Finally, there are some questions which are likely to be embarrassing; and oftentimes both patient *and* physician are glad to have them posed in a questionnaire or on an on-line system.

Some physicians have even commented that the very act of responding to a series of history questions is beneficial, for the patient is forced to think more concretely and specifically about his medical condition and is thus better prepared to answer the doctor's own questions. In other words, the patient becomes a better "historian" in terms of his ability to be a more effective participant in the doctor-patient dialogue.

Predictive information

As was pointed out earlier, the collection of medical history data is not designed to yield a diagnosis but to function as a prelude to this diagnostic process. However, there is evidence that some types of decisions can be made solely on the basis of the history data. These include the determination of the desirability of conducting certain laboratory tests or of scheduling appointments with other physicians (in a multiple-specialty group practice or hospital setting), both prior to seeing the doctor for the first time.[15]

Research

For many types of medical research, the medical record serves as an essential source of data. And for retrospective studies (reviewing past records for instances of certain occurrences), the need to have clear, complete, and readable history data can be crucial. If a researcher is looking for a possible relationship between coffee drinking and certain gastrointestinal disorders, and many of the records examined have no indication of the coffee intake, the investigation becomes that much harder.

In addition to such studies across populations, there is also the need to study individuals over time. These longitudinal studies can benefit greatly by having detailed history data, particularly review of systems information, which can both provide a health status baseline and bring to light changes over time.

By having a standardized data collection instrument, with the same question being asked of all patients in the same fashion, it becomes possible to research the data collection process itself.[16,17] Certain questions may not in fact have the sensitivity or specificity commonly attributed to them and should be replaced by others. Such new questions can be added to the system or questionnaire and then tested for validity.

Utilization reviews

In spite of the decidedly mixed reception that utilization reviews and PSRO's have received, it must be assumed that they (or something like them) will become a permanent fixture of the medical scene. Therefore, standardized histories can be helpful in enabling review committees to monitor patient care and evaluate the overall effectiveness of health care facilities.

Disadvantages

Costliness

Although it is naturally hoped that one or more of the aforementioned advantages will offset the costs, it should be clearly recognized that substantial costs will be involved. If a medical site chooses to develop its own system, there are developmental costs; and federal funds are becoming increasingly more difficult to obtain in order to help underwrite this cost. In off-line systems, with multiple-page questionnaires (some with more than 50 pages), the cost of the questionnaire design and reproduction, especially for mark sense documents, can be high. In on-line systems, the program development and the design of the terminal displays may be even higher. And in both of these approaches, the time of the physician or physicians who are guiding the project must be taken into account.

If use is made of one of the several commercially-available systems, there are still the operating costs. These can include one or more of the following: computer time, supplies, keypunching, terminal rentals and communication charges, maintenance programming (not so much to correct "bugs" as to modify and improve the questioning), and the need for attendants to administer and monitor the system.

Lack of flexibility

There can be no question but that the most flexible approach to history taking is that involving a human being as the questioner. To achieve the benefits of standardization, the corresponding cost is the loss of flexibility. If a preprinted questionnaire or prepunched deck of cards is used, a high degree of structure with regards to the questions is imposed, even though the options for administration (e.g., at home, in the waiting room, etc.) are quite flexible. On the other hand, several of the sophisticated on-line systems are quite flexible in their questioning, but are very rigid in their need for computer terminals, typically at fixed locations and limited in their ability to handle peak loads.

Lack of reliability

The more sophisticated the system, the more vulnerable to interruption it is. Whether in a batch processed mode or real-time environment, if the computer "crashes," the processing of histories stops. Where the questionnaires are keypunched or optically scanned, there is the potential for input errors. And always there is the danger lurking of the possibility of a program or system "bug." It is quite possible, of course, that a proposed new history system is every bit as reliable (or unreliable) as the traditional approach which it is designed to replace; but a new innovation must always meet a higher standard than its well-established predecessor in order to be considered a success.

Inaccuracy

In determining accuracy, the question which must be raised is "Compared to what?" It is unlikely that any system for collecting history data will ever be as accurate as the best efforts of a skilled clinician. But what of the other extreme—a careless, inexperienced, or overworked doctor?

Clearly, most present systems are not as accurate as they might be. They invariably contain false positives (reported conditions which the patient does not, in fact, have) and false negatives (the report of the absence of a condition which does actually exist). Fortunately, most systems err toward the former rather than the latter.

Wordiness

A direct outgrowth of this "overreporting," that is, the reporting of trivial or nonexistent conditions (so that real problems will not be overlooked), is that the resulting history tends to be longer than it might otherwise be. This greater length or "information overkill" (sometimes as much as 3 single-spaced pages of 8½′′ × 11′′ print-out) naturally requires more time to read and thus tends to negate the time saving benefit. Attempts have been made to streamline the output and to suppress certain non-essential data, but such efforts must be approached with great caution so that the benefits of completeness are not lost.

Impersonalness

Although less of a problem than earlier feared, the depersonalization or dehumanization of the doctor-patient encounter is a source of concern for both parties. So long as the self-administered history serves as an adjunct to the doctor's own questioning and examining, the problem should be minimal. But if it is perceived, either correctly or incorrectly, to assume a larger role, then greater resistance to its expanded use can be expected.

Confidentiality

Many physicians and civil libertarians are alarmed at the increasing loss of privacy and lack of confidentiality which is occurring with regards to the medical record. With the advent of computer-processed history data, there is a fear that this trend may accelerate. Because there are more people involved in the process than formerly (e.g., attendants, keypunchers, operators, programmers, etc.), great care must be exercised to insure that no such feared abuses occur.

Inappropriateness

When all is said and done, it must be recognized that there will always be cases where the use of a self-administered history is simply not appropriate. Such things as situations where the patient is unconscious, emergencies where action must be taken immediately, and finally patients whose educational level or literacy make it impossible for them to understand the questions—all these are instances where the use of an automated medical history system may not be appropriate.

SUMMARY

In light of the above discussion, it is easy to see that there is still much that is unresolved with regards to medical history systems. Costs are still high and many physicians are not yet convinced that there is a corresponding benefit. But as the reliability of the emerging systems improve, both from the standpoint of the hardware as well as the questions being asked, it can be expected that greater acceptance by the medical profession will follow.

REFERENCES

1. Weed, L. L., *Medical Records, Medical Education, and Patient Care,* The Press of Case Western Reserve University, Cleveland, Ohio, 1969.
2. Yarnall, S. R. and J. S. Wakefield, *Acquisition of the History Database,* (2nd edition), Medical Computer Services Association, Seattle, Washington, 1972.
3. McLean, E. R., *A Computer-Based Medical History System: Factors Affecting its Acceptance and Use by Physicians,* unpublished Ph.D. Dissertation, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1970.
4. Rockart, J. F., E. R. McLean, P. I. Hershberg and G. O. Bell, "An Automated Medical History System: Experience of the Lahey Clinic Foundation With Computer-Processed Medical Histories," *Archives of Internal Medicine,* Vol. 132, September 1973, pp. 348-358.
5. McLean, E. R. and S. Foote, "The Collection and Processing of Medical History Data: A Bibliography of Manual, Automated, and Computer-Based Techniques," *Information Systems Working Paper 4-75,* Graduate School of Management, University of California, Los Angeles, August 1974.
6. Broadman, K., A. J. Erdmann, Jr., I. Lorge and H. G. Wolff, "Cornell Medical Index: An Adjunct to Medical Interview," *The Journal of the American Medical Association,* Vol. 140, June 11, 1949, pp. 530-534.
7. Brodman, K., A. J. Van Woerkom, A. J. Erdmann, Jr. and L. S. Goldstein, "Interpretation of Symptoms with a Data-Processing Machine," *Archives of Internal Medicine,* Vol. 103, May 1959, pp. 776-782.
8. Budd, M., H. Bleich, H. Sherman and B. Reiffen, "Survey of Automated Medical History Acquisition and Administering Devices. Part I, Questionnaires," *Project Report ACP-4,* Lincoln Laboratory (M.I.T.) and Beth Israel Hospital (Harvard Medical School), Cambridge, Massachusetts, December 31, 1969.
9. Kanner, I. F., "Programmed Medical History-Taking with or without Computer," *The Journal of the American Medical Association,* Vol. 207, January 13, 1969, pp. 317-321.
10. Mayne, J. G., M. J. Martin, G. W. Morrow, Jr., R. M. Turner and B. L. Hisey, "A Health Questionnaire Based on Paper-and-Pencil Medium Individualized and Produced by Computer," *The Journal of the American Medical Association,* Vol. 208, June 16, 1969, pp. 2060-2068.
11. Collen, M. F., "Periodic Health Examinations Using an Automated Multitest Laboratory," *The Journal of the American Medical Association,* Vol. 195, March 7, 1966, pp. 830-833.
12. Slack, W. V., G. P. Hicks, C. E. Reed and L. J. Van Cura, "A Computer-Based Medical History System," *New England Journal of Medicine,* Vol. 274, January 27, 1966, pp. 194-198.
13. Grossman, J. H., G. O. Barnett, M. T. McGuire and D. B. Swedlow, "Evaluation of Computer-Acquired Patient Histories," *The Journal of the American Medical Association,* Vol. 215, February 22, 1971, pp. 1286-1291.
14. McLean, E. R., J. F. Rockart and J. H. G. Chaney, "Questionnaire Becomes Preadmission Tool," *Hospitals, Journal of the American Hospital Association,* Vol. 47, June 16, 1973, pp. 56-59.
15. Rockart, J. F., P. I. Hershberg, J. Grossman and R. Harrison, "A Symptom-Scoring Technique for Scheduling Patients in a Group Practice," *Proceedings of the IEEE,* Vol. 47, November 1969, pp. 1926-1933.
16. Collen, M. F., J. L. Cutler, A. B. Siegelaub and R. L. Cella, "Reliability of a Self-Administered Medical Questionnaire," *Archives of Internal Medicine,* Vol. 123, June 1969, pp. 664-681.
17. Hershberg, P. I., C. Englebardt, R. Harrison, J. F. Rockart and R. B. McGandy, "The Medical History Question as a Health Screening Test: An Assessment of Validity," *Archives of Internal Medicine,* Vol. 127, February 1971, pp. 266-272.

A protocol for evaluating computer systems for application in a physician's office

by DANIEL D. BUTCHER, STEVEN G. JENKS, CURTIS P. McNEELEY
Physicians Association of Clackamas County
Gladstone, Oregon

and

ROBERT E. MAHAN
Battelle
Richland, Washington

In this paper we present some of our experiences in the development of protocols for the evaluation of manual and automated medical information systems. The objective of this work has been to determine the feasibility of implementing computer technology in the office practices of private practicing physicians in Clackamas County, Oregon.

In order to effectively resolve our major objective, we established two primary tasks, namely:

- Evaluation and Documentation of Existing Manual Systems
- Evaluation and Selection of a Computer System.

These tasks, although inter-related are presented in the two sections of this paper.

BACKGROUND

On June 1, 1973, the Physicians Association of Clackamas County (P.A.C.C.) received a federal grant to determine the feasibility of cost-effective automation in the physician's office. The project, which subsequently came to be known as the Health Information Management Study (HIMS), was started on September 1, 1973. The HIMS Project is directly responsible to the P.A.C.C. Grant Committee, which in turn reports to the P.A.C.C. Board of Trustees.

P.A.C.C. was established in 1938 and is sponsored by the Clackamas County (Oregon) Medical Society. This dispersed Health Maintenance Organization (HMO) offers prepaid group coverage with service provided on a *fee-for-service* basis. In addition to a prepaid plan providing free choice of providers, P.A.C.C. has a "Protecting Circle Plan" which lists specific physicians, optometrists, physical therapists, pharmacies, medical laboratories, hospitals, convalescent hospitals, and home health care agencies providing services under the plan. These providers are scattered throughout the county which is largely rural. To minimize costs to the patient the physicians of P.A.C.C. use local control and peer review. All underwriting losses are shared by member physicians.

P.A.C.C. member physicians, recognizing that communication problems exist because of the dispersed nature of their practices, felt that an improved information system needed to be implemented. This conceptualized system would provide them access to the same kind of information they would have if they all practiced in one building with a centralized record system with easy access to one another. Such a system would improve the quality of care delivered in several ways. For example, duplication of services would be minimized. Physicians would have readily available access to information needed in a consultation or referral, thus improving physician-to-physician communication; a physician seeing a patient in a facility outside his own office would have the information available which was needed to provide the best care possible. It was felt that an automated medical information system would permit this kind of flow of information among participating physicians. With this in mind, P.A.C.C. physicians, in conjunction with Battelle, sought and received funding for such a research effort.

A major goal of this research effort was to provide an accurate comparison of current information systems and related costs to any future automated system. A format has been developed by HIMS which provides for a complete and logical description of an ambulatory medical practice. The format starts with a general description of each of the eight medical practices and includes detailed system descriptions and volume and cost statistics.

The HIMS project concerns itself with those ambulatory health care facilities located in Clackamas County and which are members of P.A.C.C. It was not feasible to document all facilities involved; therefore, a sample was selected to represent the total group. The selection criteria used in the project included the number and specialties of the physicians in the facility, the patient volume, and the proximity to other medical facilities. The sample medical practices selected represented a cross section of the popu-

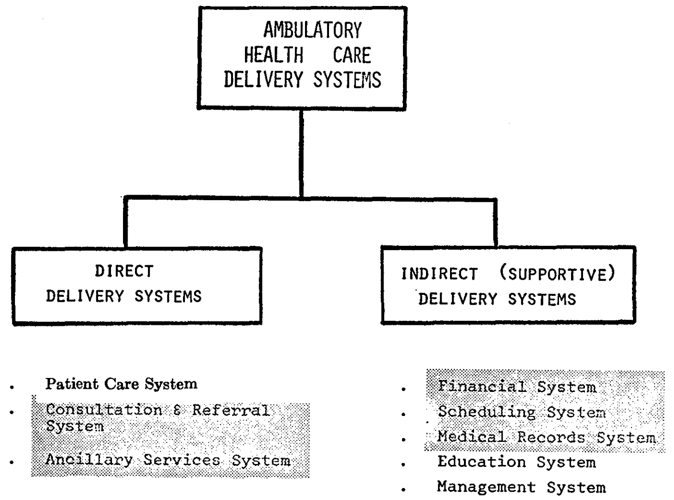

AMBULATORY HEALTH CARE DELIVERY SYSTEMS
DIRECT DELIVERY SYSTEMS
INDIRECT (SUPPORTIVE) DELIVERY SYSTEMS
Patient Care System
Consultation & Referral System
Ancillary Services System
Financial System
Scheduling System
Medical Records System
Education System
Management System

Figure 1—Health care delivery systems as found in the ambulatory health care setting (shaded area indicates those systems documented in depth)

lation which included facilities in urban and rural settings, and those which varied in size, age, and specialty.

DOCUMENTING THE MANUAL SYSTEM

Orientation and system description

The initial step in the HIMS documentation protocol was for an engineer to assume the role of a patient. This approach provided familiarization with the functions of a doctor's office and helped develop a positive rapport with the staff. While being processed as a patient through the health care facility, the engineer took notes concerning the flow of the patient and his associated information. The flow of information "triggered" by the patient was then converted to a patient-information flow diagram. This diagram, which utilized standard flow symbology, was instrumental in breaking down the medical practices into their basic "systems". Experience in Clackamas County suggests that a basic group of eight systems always exist in a medical practice. Figure 1 reflects their relationship to the total system, the medical practice.

It was our early judgment that five of these systems (shown shaded in Figure 1) were likely to be immediately affected by any automation of information. These five systems were subjected to detailed examination and documentation.

Protocol for detailed description

There are nine elements in our description of each of the five systems in each medical office:

1. System Definition and Objective
2. Flow Process Explanation
3. System Input Triggers
4. System Input
5. System Output Triggers
6. System Output
7. Physical Characteristics
8. System Problems
9. Sample of System Elements.

Figure 2 is a flow diagram of how we complete these elements and ultimately pull together an integrated system description. While content of these elements is obvious in several cases, some explanations are necessary.

Input and Output Triggers—elements 3 and 5—were *events* which caused a system to need or yield information, respectively. The input of a system was defined as the entire information received and processed by it. Somewhat differently, the output of a system is the transfer device or instrument of communication containing information produced by the system. An example of a system output is a listing of one physician's scheduled patient encounters for a given day.

The physical characteristics element included system location, storage media and preparation method of the records, accessibility, labeling, and methods used to arrange or store the records. Also considered were time relationships between information receipt, filing and retrieval,

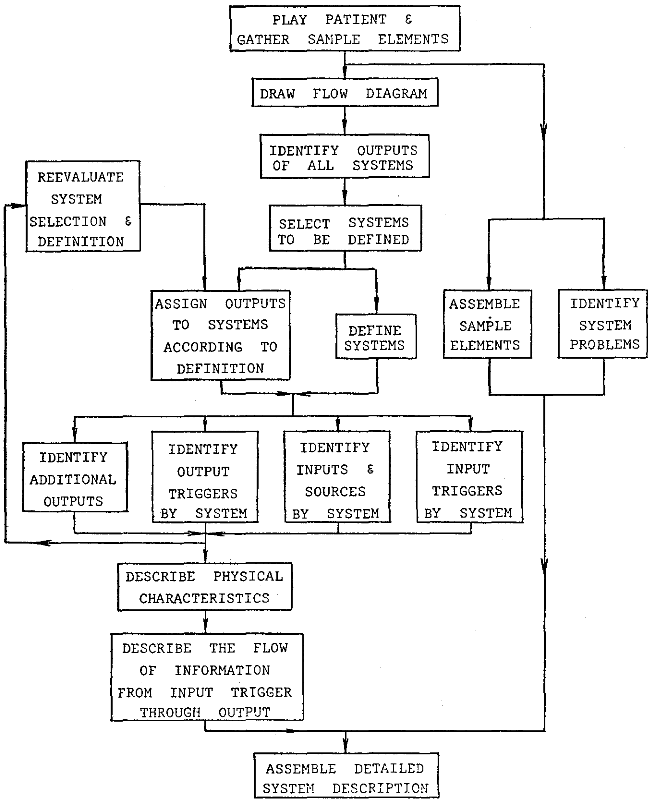

PLAY PATIENT & GATHER SAMPLE ELEMENTS
DRAW FLOW DIAGRAM
IDENTIFY OUTPUTS OF ALL SYSTEMS
REEVALUATE SYSTEM SELECTION & DEFINITION
SELECT SYSTEMS TO BE DEFINED
ASSIGN OUTPUTS TO SYSTEMS ACCORDING TO DEFINITION
DEFINE SYSTEMS
ASSEMBLE SAMPLE ELEMENTS
IDENTIFY SYSTEM PROBLEMS
IDENTIFY ADDITIONAL OUTPUTS
IDENTIFY OUTPUT TRIGGERS BY SYSTEM
IDENTIFY INPUTS & SOURCES BY SYSTEM
IDENTIFY INPUT TRIGGERS BY SYSTEM
DESCRIBE PHYSICAL CHARACTERISTICS
DESCRIBE THE FLOW OF INFORMATION FROM INPUT TRIGGER THROUGH OUTPUT
ASSEMBLE DETAILED SYSTEM DESCRIPTION

Figure 2—Flow diagram of the steps used in system identification and detailed description

and the retention criteria for a document in any given record keeping system.

A problem list of the shortcomings of a given system was made for each facility. The problems were those identified by the physician, his personnel, and the individual doing the documenting. While this list was not exhaustive in each case, the collective list of problems was of significant value while examining automation alternatives.

Measurement and analysis

Thus far in our documentation, all efforts were centered around verbal data. Hard statistical and financial data were collected to form a solid data base for future comparison. Determining the cost of an office system included allocating staff and physician time to each of the defined systems, allocating floor space to each system, and allocating all expenses according to man-hour wage, overhead, or direct assignment.

The HIMS approach to man-hour allocation involved interviews with management and staff. A spread sheet was set up for all those employed (including physicians) at the office during the year, and every effort was made to accurately assign the hours worked to the appropriate office systems encountered in an ambulatory medical facility. The physician was encouraged to log *all* of his time, including personal time, to allow a more complete analysis of his time utilization.

Once the man hour assignments were determined, a floor plan of the clinic was obtained. At this point, as much floor space as possible (usually about two-thirds of the total) was allocated to the various office systems. The remaining (unassigned) floor space, which consisted of waiting rooms, rest rooms, and hallways, was then allocated to the systems in direct proportion to the previously assigned floor space. For example, if Patient Care areas amounted to 50 percent of the directly assigned area, then 50 percent of the remaining (miscellaneous) area was assigned to this same system. Careful evaluation and a little inductive reasoning provided reasonably accurate allocations of office areas where more than one system was in operation.

The next step was to assign dollar values to the physician and staff time and floor space allocations. A copy of the annual (audited) financial report was the reference. This report reflects all expenses, and normally presents them in a detailed manner sufficient for immediate incorporation in the cost allocation effort. If an expense was attributable to one or more identifiable systems, it was appropriately charged to them. If, however, the expenses were rather broadly based (e.g., rent, fire insurance), they were assigned to the office systems in direct proportion to the assignment of floor space.

Volume statistics are equally as important as cost information when documenting any facility. Together, they provide the data necessary for evaluation of change. In pursuit of statistical data, HIMS first sought secondary sources where statistics had already been tabulated for other purposes. Since the practices we were working with did not tabulate many statistics, a plan to collect data from primary sources was then designed and carried out.

The primary objective of the collection effort was to determine the volume of patients seen during a given period. The patient medical record (including active and inactive files) was selected as the most acceptable source for these data. Information pertaining to the description of the population served was also collected at marginal additional cost. This additional information included patient characteristics (e.g., address, sex, insurance, date of birth), encounter information (e.g., date, graphics, second party involved), and report information (e.g., date, type, source).

In the HIMS effort many alternatives were considered concerning the method of sampling. It was decided that systematic sampling would be the simplest for data collectors to administer, and would not involve serious distortion of the statistics. The selection was done by counting patient records and taking every n^{th} record, with n being the inverse of the sampling rate for that office.

HIMS discovered that great care had to be taken when designing data collection forms which met the needs of both data collectors and those involved in tabulating the data. For the data collectors, the fields on the forms were arranged in the sequence that the collector would encounter the information, when going through the source document. The forms provided adequate space for writing legibility and accurate reading by tabulators.

The importance of accuracy was impressed upon the collectors, and checks on the quality of the data were conducted regularly to keep errors at a minimum. Though the sampling methods were fairly rigid, individual judgments were unfortunately made by data collectors in some instances.

The data resulting from the HIMS statistics collection effort was entered into the computer at Oregon State University. File management programs were written in FORTRAN to extract and sort data from each of the files so that continuity of information about a given patient could be maintained. No patient names were used; patient numbers were assigned to maintain data organization.

Once the cost, volume and general statistics were gathered and tabulated, there were almost infinite possibilities for analysis. However, the present needs were for system costs and for several basic volume and growth statistics. All the collected data is stored on magnetic tape for possible future use.

Discussion of problems

Probably the most serious problem facing the Industrial Engineers working on this project was the obvious meaning of "cottage industry". There are approximately 140 physicians in Clackamas County working out of nearly seventy offices. The largest clinic in the county presently is made up of five physicians of various specialties. Thus, the size alone of these practices suggests the problem of a lack of internal documentation and standardization. While many similarities were found when comparing systems of different offices, many more differences were documented.

As would also be suggested by the relatively small size of these practices, statistics are neither needed nor maintained. Thus, the engineers were working in a "virgin" territory, relatively untouched by computers, systems experts, or professional managers.

Faced with this environment, the HIMS staff carried out plans to "soften" the blow of systematic analysis in these offices. With the obvious long range goal of automation clearly understood by the office staffs, the age old operational problems of the "efficiency expert" were anticipated. Meetings were held with over sixty office personnel which thoroughly explained and discussed HIMS. The net result was a definite "supportive" attitude that has persisted throughout this study.

The extensive physician involvement in this study was somewhat unique. This fact relates to the history of this study, which has the unique aspect of a group of physicians seeking out technical assistance in the area of automation, instead of the far more common reverse situation. Truly, the HIMS project represents a combined medical/engineering approach to problem solving in the medical environment.

While many other problems have been faced, one very interesting one has occurred, against which no known cure exists. Members of the medical profession are, at times, impulsive. This trait, combined with the intensive, deliberate exposure to automation occasioned by HIMS has resulted in "overpriming" some physicians. Several problems occur as a result of this situation. Examples experienced by HIMS include efforts by one physician to have a local computer service organization obtain a franchise for one system he had seen, since he decided unilaterally that system he had seen, was for him. He felt HIMS would take too long to study the offices before implementing a system. Another difficulty is the waiving of interest and support that occurs when the engineers finish documenting one office and move into another. This "withdrawal" of interest is not easy to restore, and can breed discontent throughout a medical group. HIMS practice to date has been to seek a careful balance between educating the physician in the applications of computers, necessary to obtain his judgment on potential value of the new system, and not educating him, which tends to alienate him and cause negative support.

Summary

The protocol presented in this paper can and has provided a comprehensive picture of private medical practices. It functions best in a cooperative mode, with the benefits being a mutual education of medical and systems personnel alike. Some traditional systems analysis problems do occur, but our experience indicates that they tend to be soluble to a high degree, possibly due to the higher than average education level of the personnel involved.

COMPUTER SYSTEM SELECTION

Selection constraints

Before proceeding to the topic of developing a system selection protocol, we should point out some of the constraints that we imposed on ourselves during the program. The most significant was the requirement that the system be largely composed of off-the-shelf components, especially the system software. We have been and continue to be far more interested in utilizing the best of available hardware and software than we are in developing new innovations to be tested. As a second constraint we elected to pursue an incremental, or modular approach to implementation as has been suggested by Dr. G. O. Barnett.[1] Our purpose in this case was to provide a basic system which could be implemented at a reasonable risk (as opposed to a "total" system), but one which would provide a reasonable degree of open-endedness so that it could be substantially expanded. Third, we set out purposefully to involve physicians and physicians' office staff in every major (and many minor) analysis, design, and implementation decisions. Since the ultimate acceptance or rejection of the system will rest on choices made by these people it is not just prudent, but necessary, to learn how to utilize their skills in assisting with the selection process.

Selection methodology

The methodology (protocol) developed for use in selecting a computer system is based on comparing the performance of modern medical computer systems against the requirements developed for P.A.C.C. The protocol includes four major elements which are:

- Definition of System Performance Measures
- Analysis and Specification of Requirements (Modeling)
- Performance Analysis of Available Systems
- System Selection.

This methodology is broadly comparable to that surveyed recently by Timmreck.[2] He defines the four steps as (1) analysis and specification of need, (2) request for proposals, (3) validation of proposals including system performance measurement and (4) actual system selection. It may be contrasted with selection criteria reported by Raymond[3] and Giebink.[4] There are several major differences between our methodology and most others.

First, the system performance measures have been defined for a broad class of medical computing systems, namely real-time systems suitable for a geographically diverse group of physicians in a combination rural and suburban setting.

The second difference in this methodology is that the definition of system performance measurement elements are presented from several different viewpoints; namely, the user, the computer scientist, and the business manager. The advantage to the user of this type of protocol is that he can develop an understanding beyond the technical performance of the system. Additional information gained includes a measure of the type (or style) of vendor he is dealing with, for the user's acceptance of the system, for its expansion capability, and the many other factors that are a part of the total commitment involved in owning or using a computer system for medical applications.

Third, we should stress that the selection process is not a simple four step process. It is a recursive process that must be repeated and refined. While we consider the basic four steps to be essential to every computer selection, we also recognize that each step can be approached in a simple manner or in a great deal of depth. We cannot, for example, tell every user that they should examine their requirements in the depth that we have at P.A.C.C. It would not, for example, make any sense to spend $50,000 selecting a $20,000 system. We can suggest that performance measurement elements and a system model should always be established prior to any commitment to lease or buy either a computer system or service. It is left to the individual user to determine the depth of investigation and analysis appropriate to his particular needs.

System performance measures

Performance measures are a set of measurable elements that characterize the system being considered. Performance measures should not be confused with performance specifications. By defining a set of measures we are simply generating a glossary of elements that are indicators of the performance of computer systems. Specifications are specific performance criteria which are used to describe the desired system in terms of the measurement elements.

The performance measurement elements derived for P.A.C.C. are shown in detail in Appendix A of this report. Six major categories of measurement have been selected. They are:

- Vendor Overview
- Systems Performance
- Applications Performance
- Start-up and Expansion Capability
- User Interface Performance
- Cost.

Vendor overview is intended to provide background information on each prospective system supplier. We have not set specific standards of acceptance in this area, but expect the information to be useful to P.A.C.C. in becoming familiar with each specific company. Information collected for each vendor includes:

- Corporate Profile
- Product Profile
- Health Care Experience.

Systems performance measurements are indicators of the fundamental design capability of the system. The use of evaluation criteria in this area can be eliminated or limited to a subset of the elements presented in some selection processes. This is especially true for a medical group that is selecting a system to solve a specific, well-defined problem such as taking patient-histories. In such a case, the evaluation may be carried out based on the quality of the applications programs, the user interface, cost, etc., without regard for the design of the system. System design is an important element in any comprehensive information system, especially one that will have a county-wide impact as contemplated at P.A.C.C. Comprehensive, multiple-user systems are complex dynamic systems which must be flexible in design and operation, easy to maintain, and provide the necessary hardware and software support so that they can be modified to meet new and expanding needs. If existing systems are so designed that they do not meet the needs of the specific medical community where the system is to be used then the basic system design is a critical factor and should be considered. If the application is fairly simple and well proven then a prospective user of the system does not have to be so cautious.

Systems performance is characterized by the basic design of the hardware and the system software available to the user. The major elements are:

- Hardware Capability
- Programming Languages
- Operating System
- Data Base Management
- Utility (Support) Programs
- System Upgradability
- System Support
- Stage of Development.

Applications programs are the elements of the computer system that actually solve the end user's problems. It is the applications program that must be most closely tailored to the specific user's needs and demands. Many examples of computer system failures can be directly traced to applications programs which solved the wrong problem or solved the right problem in an inferior manner (e.g., at an excessive cost for the task). It is of vital importance that those who design the system (the computer types) and those who use the systems (physicians, nurses, business staff) work together in the analysis of application requirements, design, and use. It is extremely helpful to have access to a physician with a background in computers to act as the intermediary for these two groups.

Examination of available applications should focus on the scope, or breadth, of applications available and on the specific performance of applications packages. The following classes of applications are being implemented or considered at P.A.C.C.:

- Practice Management
- Business

- Health Care (Medical)
- Insurance
- Communications (physician-physician, etc.).

Start-up and expansion capability is of prime concern to the prospective user of a medical computing system. He must develop a plan for the effective implementation of a proposed system. It is desirable for the user to minimize his risk and capital expenditure by selecting the minimum hardware/software system that can be implemented to solve his current problems. At the same time, the system must be large enough to be capable of expansion to meet long range growth (both in the number of users, in the various types of applications programs, and in the population served). It can be as much of a mistake to install a large system which is grossly under utilized as it is to install a small system that cannot be expanded to meet all of the important design needs.

We are suggesting that the best approach is one that permits the modular expansion of the hardware/software system as the capabilities of the system expand. Our approach to this problem is to model both the full scale system and a start up system where the full scale system will meet the needs of our total user group (the physicians of Clackamas County, Oregon), but the start up system will begin with a small group of physician offices (say 7 or 8).

The growth and acceptance of medical computing systems depends largely on the establishment of effective man-machine communications. In a larger sense, the success of the system depends on the performance of the user interface with the system whether that involves a user at a terminal, or a user attempting to change the system (via programming or administrative action), or a user attempting to understand the system (training and documentation). The performance of the system from the user's viewpoint can be measured in terms of (1) What is used (hardware), (2) How it is used (communication/display elements), (3) How information is protected (security), (4) When it can be used (hours of service), (5) How well people like it (human factors design), and (6) How well it is documented. The performance elements are listed as follows:

- User Equipment (Hardware)
- Communication/Display Elements
- Security
- Hours of Service
- Human Factors Design
- Documentation and Training.

The cost of a medical computing system should be treated very carefully by the prospective buyer. Computing systems are not like conventional instruments such as X-ray machines and the like. They are complex and dynamic systems that require long term investments in personnel and expansion. Costs will vary substantially based on whether the user elects to purchase a computer service rather than installing his own system. In either case, estimates should be made for one-time costs and for the costs of recurring items. Design, implementation, testing, operation, and maintenance costs should also be calculated. Both personnel and facility costs should be added which will include terminal operators, central computer operators, programming staff (if any), management, and maintenance. Facility costs should not only include the facility to house the computer, but required improvements (power, air conditioning, etc.) and a cost associated with the space required for the terminal in each user's office.

Analysis and specification of requirements

Having defined the system performance elements we next begin to define performance requirements for each element. These requirements then become our model for the desired system. It is very useful to define two types of requirements: mandatory and desirable.[5] In our case, we were able to reduce our analysis of vendor systems from a large number to only a few by the mandatory requirement that the system perform its major tasks on a real-time basis.

Requirements were established largely by the P.A.C.C. Grant Committee working with HIMS staff engineers. Inputs to the effort included (1) data from the documentation effort described earlier in this paper, (2) observations of office practices and problems by HIMS staff engineers performing the documentation effort, (3) a comprehensive Delphi Survey[6] of physicians to determine their perceptions of information processing problems and their respective priorities, (4) a similar Delphi Survey of nearly 300 physician employees, (5) site visits to a number of medical computer installations, (6) limited evaluations of several systems that were demonstrated on a limited scale at our site, (7) a review of vendor literature, and (8) review of the current open literature concerning medical computing systems.

The results of the Delphi indicated that both physicians and their employees consider insurance claims processing, private billing, and internal control of both medical and financial information to be of highest concern. To a lesser degree physicians were concerned with inter-office communications. There was very little perceived need for some of the classical medical computing applications. The results of this study coupled with the documentation has led us to the following conclusion. The source of the underlying operational problems of office practice can be largely attributed to the current structure of the medical and financial records system. Moreover, if the data contained in these records were organized and stored in a manner that promoted effective retrieval and use of the information required for (1) insurance and private billing, (2) medical care, and (3) physician-to-physician communications then many of the current (and costly) redundancies of personnel and services could be reduced.

The next major task undertaken was to translate these conclusions into computer system requirements. The P.A.C.C. Grant Committee, HIMS Staff, and Battelle began a series of workshop meetings that were designed to develop a set of requirements in terms of the performance measures previously developed. We found that it was relatively easy at this point to develop criteria (standards) for selection since some of the more difficult problems had already been solved. A consensus had been established by the Delphi Study. The documentation effort supported the consensus with hard data and provided the means by which the general problems identified in the Delphi Study were linked to specific office systems and protocols. The development of performance measures provided an educational opportunity to familiarize physicians with the characteristics of medical computing systems before the selection process began. And finally, the interaction process provided the time needed for a group of broadly diversified people to learn how to effectively communicate with one another.

Performance evaluation

Performance evaluation of currently available systems requires three steps:

- Identification of Systems
- Analysis of Systems
- Selection or Evaluation.

One of our continuing efforts has been to identify candidate systems for evaluation. A variety of techniques were used, some successful, some not. The least successful was the preparation of a letter to vendors describing our general needs. The most successful were references from people in the industry and identification in the open literature. Initial identification was followed by telelphone inquiry and if the system appeared suitable, by a site visit. At this point we established a screening process against our mandatory criteria. The following requirements were used:

- The system must be primarily a real-time system. Some applications (such as billing) can be run off-shift, but data entry must be on-line.
- The system must be capable of supporting up to 100 physicians in a time-sharing mode and in a diverse geographic environment
- The system must have a background of established performance
- The system must provide applications software
- The applications must support both business and medical functions.

Only four systems survived the initial screening process. The secondary screening process utilized the detailed standards developed by the HIMS staff, the Grant Committee, and Battelle. Side-by-side comparisons were made for each system which resulted in the elimination of two more systems. In order to again refine the process, the final two vendors were asked to submit proposals for consideration. In addition, the Grant Committee reviewed the requirements and established the four most important elements which are listed as follows in descending order of importance:

- Cost
- Scope of Business and Medical Applications
- Human Factors Design
- Hours of Computer Coverage.

It should be pointed out that we did not systematically develop a weighting scheme for evaluation, largely because of the difficulty in assigning weights to all of the elements. Rather, we used the workshop meetings to develop a relative weighting scheme for selected elements based on interaction between the workshop participants.

The final evaluation which resulted in the selection of a preferred system consisted of the repeat of the side-by-side comparison of each system plus the development and presentation of a complete set of cost breakdown sheets for each system which included a broad variety of lease, purchase, depreciation choices.

SUMMARY

We have presented a protocol for the systematic analysis and selection of a computer system for a diverse medical community. The process involves physicians, physician employees, industrial engineers, and computer scientists working in a collaborative atmosphere. As a result, a large sample of the medical community of Clackamas County is prepared to implement a computer based information system in 1975.

REFERENCES

1. Barnett, G. O., "Massachusetts General Hospital Computer System," Chapter 17 in *Hospital Computer Systems,* Morris Collen (ed.), John Wiley and Sons, New York, 1974, pp. 517-545.
2. Timmreck, E. M., "Computer Selection Methodology," *Computing Surveys,* Vol. 5, No. 4, pp. 199-221, December 1973.
3. Raymond, S., "Criteria in the Choice of a Computer System," Parts I and II, *Journal of the American Medical Association,* Vol. 228, Nos. 5 and 8, pp. 591-594 and 1015-1017, April 29 and May 20, 1974.
4. Giebink, G., *Computer Selection for a Patient Information System,* Health Data Management Systems, Denver, Colorado. February 1972.
5. Joslin, Edward O., *Computer Selection,* Addison-Wesley, Reading, MA, 1968.
6. Health Information Management Study, *Progress Report Volume I and II,* March 31, 1974.

APPENDIX A—SUMMARY OF TECHNOLOGICAL/COST EVALUATION ELEMENTS

I. *VENDOR OVERVIEW*
  A. *Corporate Profile*
    1. Parent Company
    2. Data Founded
    3. Number of Employees
    4. Employee Skill Mix
    5. Number of Active Customers
    6. Length of Customer Service
    7. Profitability
    8. Sales Volume
  B. *Product Profile*
    1. Type of Product(s)
    2. Completeness of Product Line
    3. Market Segment Serviced
  C. *Health Care Experience*
    1. Installed Systems
    2. Staff Experience
    3. Source of Medical Input
II. *SYSTEMS PERFORMANCE*
  A. *Hardware Performance*
    1. Type of Hardware
    2. Configuration
    3. Central Processor
    4. Memory
    5. Major Peripherals
    6. Data Communications
  B. *Programming Languages*
    1. Language Types
    2. Programming Ease
    3. Standardization
    4. Portability
    5. Efficiency
  C. *Operating System*
    1. Type of System
    2. Scheduling
    3. Number of Tasks/Partitions
    4. System Compatibility
    5. Security
  D. *Data Base Management*
    1. Type of System
    2. File Structure
    3. Access Techniques
    4. Security
  E. *Utility (Support) Programs*
    1. Diagnostics
    2. Program Development Tools
    3. Other Available Programs
  F. *System Upgradability*
    1. Age
    2. Production Status
    3. Upward Compatibility
    4. Ease of Program Conversion
  G. *System Support*
    1. Vendor Support
    2. Maintenance Requirements
    3. Operation Requirements
    4. Space Requirements
    5. Power and Cooling Requirements
  H. *Stage of Development*
III. *APPLICATIONS PERFORMANCE*
  A. *Practice Management*
    1. Statistics Reporting
    2. Peer Review
    3. Budgeting
  B. *Business*
    1. Aged Accounts Receivable
    2. Daily Transactions
    3. Post Accounts (Close Books)
    4. Trial Balance
    5. General Ledger
    6. Billing
    7. Payables
    8. Dictionaries
      a. Consumer
      b. Vendor
    9. Subsidiary/Special Accounts
    10. Edit (all other applications)
  C. *Health Care*
    1. Medical Records
    2. Scheduling
    3. Patient Histories
    4. ECG Analysis
    5. Diagnosis
    6. Laboratory
    7. Pharmacy
    8. Drug Interaction
    9. Training/Education
  D. *Insurance*
    1. Third Party Claims
    2. Third Party Receipts
    3. Claim/Receipt Justification
  E. *Communications*
    1. Message Switching
    2. Transferring Files
    3. Accessing File Subsets
IV. *START-UP AND EXPANSION CAPABILITY*
  A. *Minimum Break-even Configuration*
  B. *Expansion Capability*
V. *USER INTERFACE PERFORMANCE*
  A. *User Equipment*
    1. Input Devices
    2. Output Devices
    3. Miscellaneous Devices
  B. *Communication/Display Techniques*
    1. Interaction Protocol
    2. Response Time
    3. Error Control
    4. System Failure Recovery
  C. *Security*
    1. Responsibility
    2. Security Schemes
    3. Violation Detection
    4. Audit and Reporting

D. *Hours of Service*
 1. Daily
 2. Nights
 3. Weekends

E. *Human Factors Design*
 1. Operator Acceptance
 2. Language Requirements (Special Encoding)
 3. Training Time
 4. Space Requirements
 5. Convenience of Use
 6. Office Disturbance

F. *Documentation and Training*
 1. Systems
 2. Applications
 3. User

VI. *COST*

A. *Time Shared Access*

B. *Owning the System*

A clinical information system (CIS) for ambulatory care

by CLEMENT McDONALD, BHARAT BHARGAVA and DAVID JERIS
Regenstrief Institute
Indianapolis, Indiana

INTRODUCTION

In this paper we wish to present an evolving ambulatory care information system (CIS) which has been in use by physicians since July 1, 1973. The noteworthy features of this system include:

1. Implementation in a high level interpretive language on a mini computer.
2. Paper based (optically read turn-around documents) rather than electronic terminal I/O.
3. Automated clinical surveillance (the computer works for the physician by searching out errors and danger conditions rather than being a passive repository of data).
4. Data base management orientation to better cope with the constant companion of change.

By ambulatory care we mean medical care provided in physician offices and outpatient clinics. We call the place where such care is given a Care Facility (CF). It might be noted that our host facility is part of a hospital—and to some extent the CIS provides service to the hospital as well as the outpatient clinic.

In the discussion which follows we distinguish between legislative and executive decisions. The former is a decision to change operational rules. The latter decision is a decision to act based on pre-defined operational rules. The purpose of the CIS to be described is to do executive work at each level: the clinical, ancillary and administrative level, and to enlighten legislative decisions by providing a statistical overview of the system's operation. Because of our emphasis on ambulatory care, the clinical service modules do not serve inpatient care, the ancillary service modules do.

THE CLINICAL INFORMATION SYSTEM

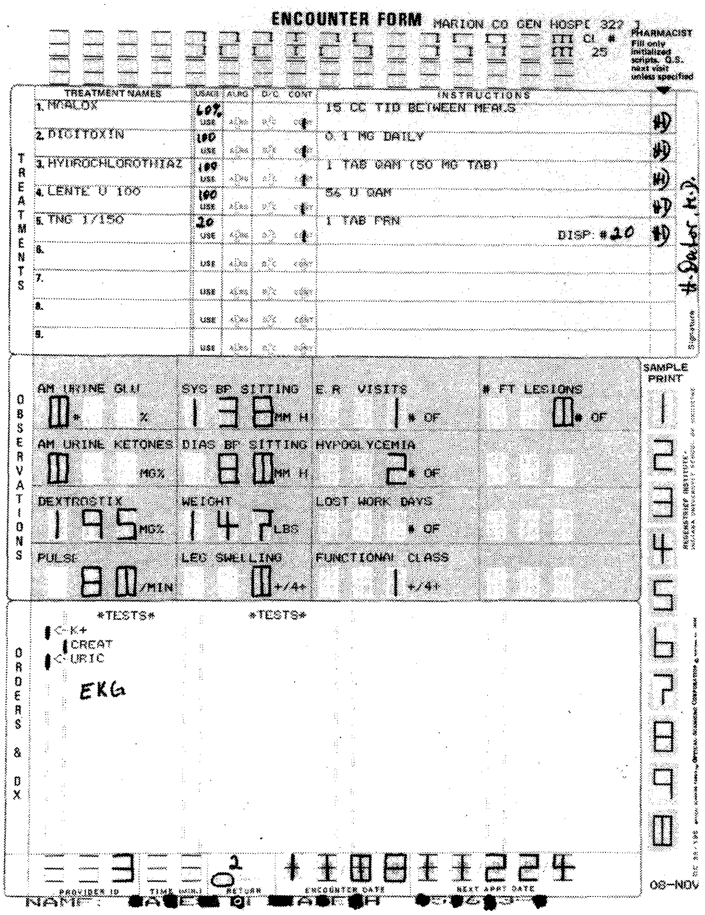
System overview

The CIS is designed to deal with a large, 200,000 active patient population at the administrative-ancillary service level and a smaller 40,000 patient subset of that population at the clinical level. Our limits are based on both mass storage considerations and the statistical realities of our hospital outpatient facility. The largest DEC supported mass storage device holds 43 million words or approximately 168,000 sectors. Ancillary services and administration require access to a fixed minimum of patient identification data. Patient registration requires approximately 100 bytes of storage per patient (depending upon the CF's needs, more or less storage could be required). Our target of 200,000 patients fits easily and occupies 40,000 sectors; larger populations would be feasible but are not necessary to our environment. Based on our projected average medical record size of 3 sectors, 120,000 sectors allows for 40,000 patients. This limit seems severe in proportion to the size of active registry, however, all clinics do not have equal need for full clinical services. By propitious shifting of patients from active to inactive storage, these limits can be softened. We have not yet taken delivery of our large mass storage device and to date have not accumulated any experience with very large files. Thus we are not certain that the system resources will be able to support all of the activities we have planned for such a large patient population.

Operational elements of an ambulatory care facility

An ambulatory care facility deals with four populations of elements: patients, physicians, clinic and parameters. These elements must be duly registered in the CIS by name and necessary descriptive characteristics. For example, when a patient is registered, we record his name, hospital number, date of birth, race, sex and other characteristics. The data base approach of the CIS allows flexibility in the choice and the number of such attributes.

The notion of a parameter and what should be registered about a parameter may not be obvious. By a parameter we mean any clinical observable, including treatments, tests, physical findings, items of information from the patient's history, and second order aggregates of preceding elements. Thus, "penicillin" is a parameter, "electrolytes" is a parameter which is second order because it subsumes four primitive parameters: sodium, potassium, chloride and bicarbonate. The parameter set which describes a clinical universe varies from CF to CF. It represents the discrete descriptive approximation to a



ENCOUNTER FORM MARION CO GEN HOSPC 322 J

CL # 25

PHARMACIST Fill only initialized scripts. Q.S. next visit unless specified

TREATMENTS

	TREATMENT NAMES	USAGE	ALRG	D/C	CONT	INSTRUCTIONS	
1.	MAALOX	60% USE	ALRG	D/C	CONT	15 CC TID BETWEEN MEALS	HJ
2.	DIGITOXIN	100 USE	ALRG	D/C	CONT	0.1 MG DAILY	HJ
3.	HYDROCHLOROTHIAZ	100 USE	ALRG	D/C	CONT	1 TAB QAM (50 MG TAB)	HJ
4.	LENTE U 100	100 USE	ALRG	D/C	CONT	56 U QAM	HJ
5.	TNG 1/150	20 USE	ALRG	D/C	CONT	1 TAB PRN DISP: #20	HJ
6.		USE	ALRG	D/C	CONT		
7.		USE	ALRG	D/C	CONT		
8.		USE	ALRG	D/C	CONT		
9.		USE	ALRG	D/C	CONT		

Signature H. Baker, M.D.

OBSERVATIONS

AM URINE GLU 0 %	SYS BP SITTING 138 MM H	E.R. VISITS 1 # OF	# FT LESIONS 0 # OF
AM URINE KETONES 0 MG%	DIAS BP SITTING 80 MM H	HYPOGLYCEMIA 2 # OF	
DEXTROSTIX 195 MG%	WEIGHT 147 LBS	LOST WORK DAYS # OF	
PULSE 80 /MIN	LEG SWELLING 0 +/4+	FUNCTIONAL CLASS 1 +/4+	

SAMPLE PRINT 1 2 3 4 5 6 7 8 9 0

ORDERS & DX

TESTS *TESTS*

<-K+
CREAT
<-URIC

EKG

PROVIDER ID	TIME (MIN)	RETURN	ENCOUNTER DATE	NEXT APPT DATE
3		0 2	1 1 0 8 1 1	2 2 4

05-NOV

NAME: [redacted]

Figure 1

continuous universe. For each parameter, its name, synonyms, type (whether tree, scalar, etc.), normal range, absolute range, units, and certain other characteristics must be registered before observations about that parameter can be accepted.

The clinical information system and the care facility

The CIS is designed to serve the CF at three levels: clinical, ancillary and administrative. However, the main impact of CIS on the quality and cost of care is made by the services it provides at the clinical level. The clinical level also involves the most severe problems, hence, the emphasis is on the clinical level in this report.

The inputs to the clinical record include all patient related information. This includes information gathered by the physician, by the nurse, by the lab, by radiology, and by other ancillary services.

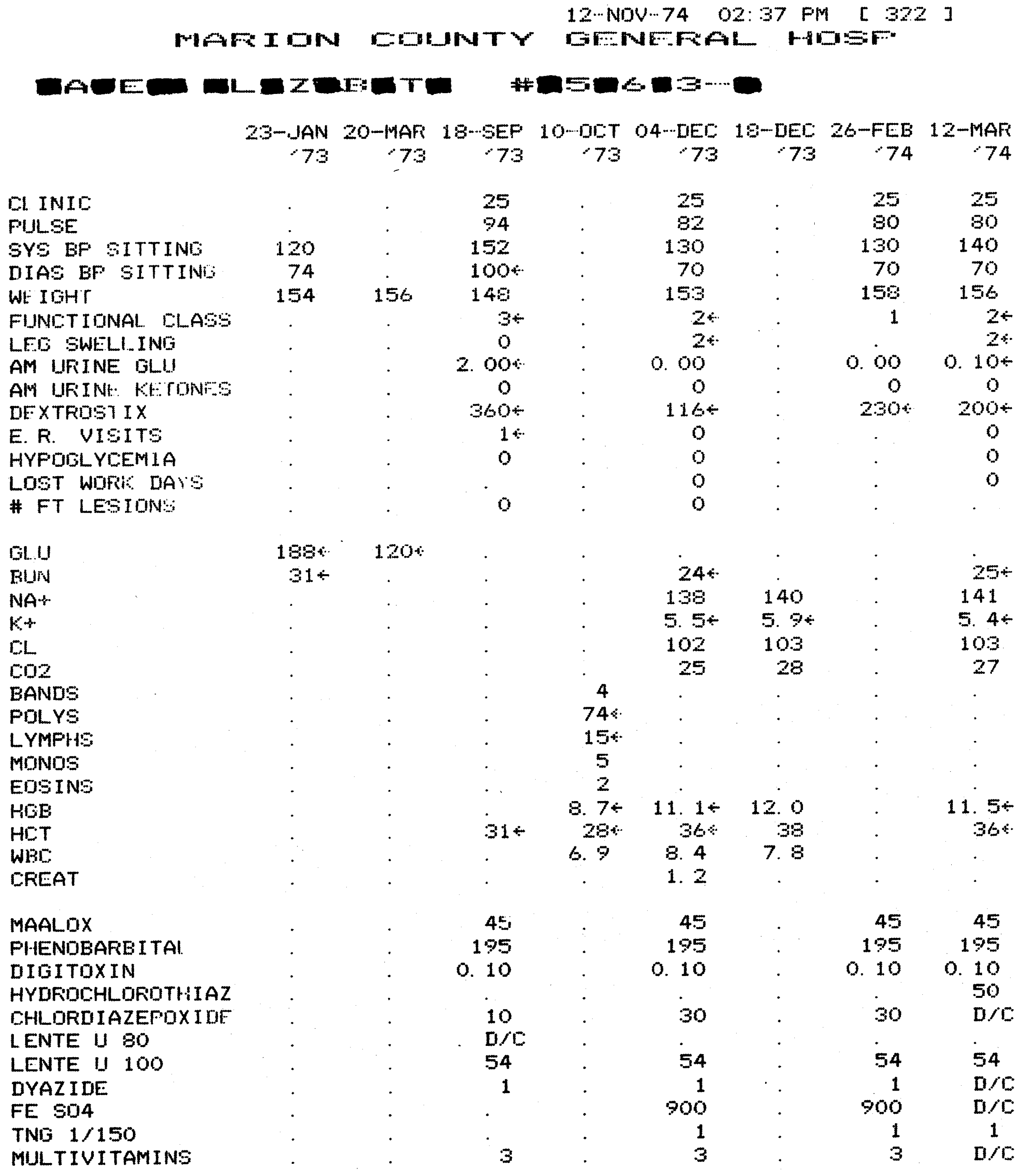
One must view separately the problems of trapping ancillary service data from those of physician acquired data. Ancillary service data can be trapped in two modes. The most efficient is via a computerized ancillary module. The alternative is for a clinic based clerk to enter such data either via optically read forms tailored to the statistical distribution of test usage or via direct terminal entry.

How to trap data produced by the physician during the clinical encounter? This is the thorniest problem. Grossman, et al. have developed an elegant approach involving standardized encounter forms and dictaphone transcriptions.[1] We have taken a different tack. First we assume that most data trapping within the clinical environment will be expensive since it involves in one way or another an expensive resource—the physician. Second, we assume that most of the data he collects is not of sufficient archival importance to justify high costs (for data of short term utility, his handwritten notes will suffice). Given these two assumptions, we let him decide what parameters are of archival importance to him and ask him to record observations for these on our optically read forms. He must distinguish between two classes of importance: conditional or unconditional. In a general medicine clinic, the urine glucose might be designated as a conditionally important parameter under the condition that the patient is diabetic. Conversely, blood pressure would be declared unconditionally important given the proven consequences of hypertension and our ability to reverse them with treatment. Parameters that are of declared importance to the clinician appear on the encounter form (Figure 1). Those that are unconditionally important always appear. Those that are conditionally important appear only when the specified conditions are met. The CARE language to be discussed later provides the mechanism for specifying conditional relationships which vest importance on a given parameter. As one may notice from Figure 1, space limitations constrain us to a maximum of sixteen important parameters on any one patient. This has not been confining for our outpatient environment.

For more detailed data input by the physician, two mechanisms are available. First, special turn-around documents can be tailored to an individual physician's data input needs. As many as sixty different parameters can be specified on a single form. Second, keyboard terminal entry is available. In addition, we are now developing a tree structured multiple choice form which can also be produced as a turn-around document. Our underlying bias is that much of the voluminous free text data found in the conventional medical record is not of sufficient archival value for computer storage, and thus these special input mechanisms will only infrequently be required.
response to a patient visit in anticipation of a return. In this second case, there must be a delay between the patient encounter and report production to assure that outstanding test requests have been completed and returned to the system.

Clinical reports

The first report is the summary report (Figure 2). This report is a flow sheet displaying the time course of all recorded parameters. Physical findings, historical data,



```
                                         12-NOV-74   02:37 PM   [ 322 ]
              MARION     COUNTY     GENERAL     HOSP

[redacted]A[redacted]E[redacted] [redacted]L[redacted]Z[redacted]B[redacted]T[redacted]     #[redacted]5[redacted]6[redacted]3--[redacted]

                       23-JAN  20-MAR  18-SEP  10-OCT  04-DEC  18-DEC  26-FEB  12-MAR
                         '73     '73     '73     '73     '73     '73     '74     '74

CLINIC                    .       .      25       .      25       .      25      25
PULSE                     .       .      94       .      82       .      80      80
SYS BP SITTING          120       .     152       .     130       .     130     140
DIAS BP SITTING          74       .     100←      .      70       .      70      70
WEIGHT                  154     156     148       .     153       .     158     156
FUNCTIONAL CLASS          .       .       3←      .       2←      .       1       2←
LEG SWELLING              .       .       0       .       2←      .       .       2←
AM URINE GLU              .       .    2. 00←     .    0. 00      .    0. 00   0. 10←
AM URINE KETONES          .       .       0       .       0       .       0       0
DEXTROSTIX                .       .     360←      .     116←      .     230←    200←
E. R.  VISITS             .       .       1←      .       0       .       .       0
HYPOGLYCEMIA              .       .       0       .       0       .       .       0
LOST WORK DAYS            .       .       .       .       0       .       .       0
# FT LESIONS              .       .       0       .       0       .       .       .

GLU                     188←    120←      .       .       .       .       .       .
BUN                      31←      .       .       .      24←      .       .      25←
NA+                       .       .       .       .     138     140       .     141
K+                        .       .       .       .     5. 5←   5. 9←     .     5. 4←
CL                        .       .       .       .     102     103       .     103
CO2                       .       .       .       .      25      28       .      27
BANDS                     .       .       .       4       .       .       .       .
POLYS                     .       .       .      74←      .       .       .       .
LYMPHS                    .       .       .      15←      .       .       .       .
MONOS                     .       .       .       5       .       .       .       .
EOSINS                    .       .       .       2       .       .       .       .
HGB                       .       .       .     8. 7←  11. 1←  12. 0      .    11. 5←
HCT                       .       .      31←     28←     36←     38       .      36←
WBC                       .       .       .     6. 9    8. 4    7. 8      .       .
CREAT                     .       .       .       .     1. 2      .       .       .

MAALOX                    .       .      45       .      45       .      45      45
PHENOBARBITAL             .       .     195       .     195       .     195     195
DIGITOXIN                 .       .    0. 10      .    0. 10      .    0. 10   0. 10
HYDROCHLOROTHIAZ          .       .       .       .       .       .       .      50
CHLORDIAZEPOXIDE          .       .      10       .      30       .      30     D/C
LENTE U 80                .       .     D/C       .       .       .       .       .
LENTE U 100               .       .      54       .      54       .      54      54
DYAZIDE                   .       .       1       .       1       .       1     D/C
FE SO4                    .       .       .       .     900       .     900     D/C
TNG 1/150                 .       .       .       .       1       .       1       1
MULTIVITAMINS             .       .       3       .       3       .       3     D/C
```

Figure 2

treatments, lab tests, x-rays and other ancillary service measurements are all represented. Abnormal numerical results are flagged by arrows. In order to preserve the matrix format, tree valued parameters are represented by a two letter code which by convention is stored at the root node of every tree result. Both the state and trend are represented. The state is described as normal (N), pathologic (P), or different (D). The last category indicates minor abnormalities or findings which are normal variants. The trend is indicated as better (B), worse (W) or same (S).

The value of a flow sheet lies in its clear display of data over time. Change and the rate of change, both of which are crucial determinants in medical decisions, are readily apparent. In addition to the standard total summary, the system provides summaries restricted to a specific time window or a specified set of parameters, the abnormal report which displays only abnormal parameters, the compact report by which parameter trajectories for multiple patients are displayed on a single page. This latter report supports clinical legislative decisions by presenting data about a large number of patients in a compact form. Any of the above formats can be displayed on a CRT terminal in real time or printed on paper in batch mode.

The second report referred to above is the encounter form. We have already alluded to this report which is a two-part turn-around document with many purposes. It supports the physician's assimilation of the patient's active treatment states by displaying the patient's active treatment profile at the top of the form (Figure 1). Notice that the profile is actually a list of prescriptions as written by the physician. This is not by coincidence, since the encounter form is a constituent of the prescription process as we will see later.

The CARE language

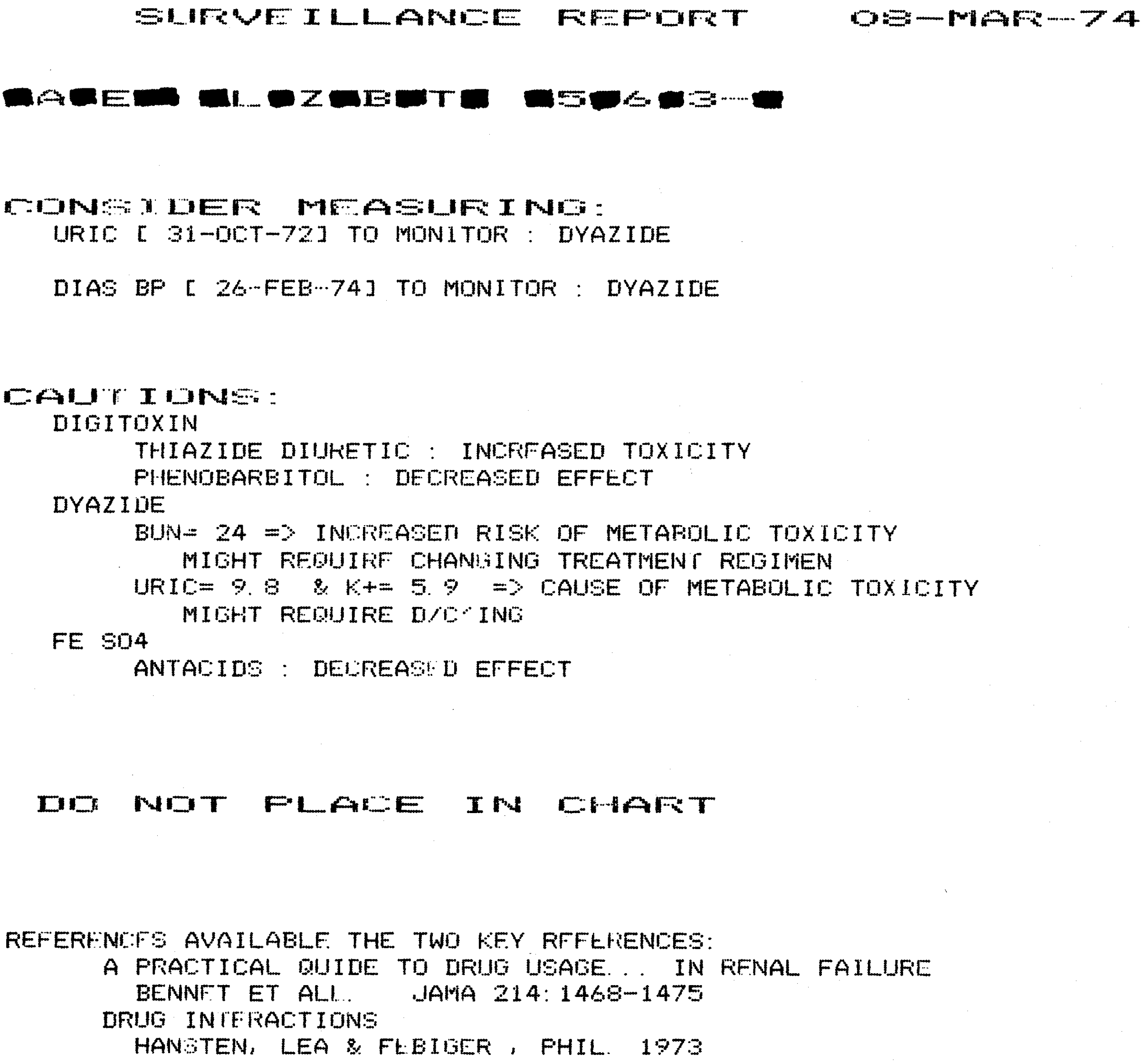
Before discussing the third report provided as input to the clinical level, it is necessary to discuss the language by which the physician defines his executive rules. The CARE language provides us with two capabilities, one which helps legislative decisions, the other executive decisions. The language is interpreted by a keyword driven compiler written in BASIC-PLUS. The CARE language allows the user to specify clinical conditions and consequent therapeutic or diagnostic actions in terms of if-then-else constructs. The building block of the language is the parameter clause containing a parameter subject and one or more modifiers. There are two kinds of clauses, action clauses and conditional clauses. For conditional clauses, the modifiers express restrictions on the temporal course of the parameter and the range of magnitudes or changes in magnitudes. Conditional clauses are linked by the Boolean operators "and", "or", in addition to temporal connectives "and followed by", "and preceded by" "and coincident with". "Then" and "else" can be used to relate conditional clauses to action clauses. The action clause includes a suggestion to do something in relation to the subject parameter and the justification for that suggestion. For example, "order urine glucose because of risk of metabolic toxicity" is a valid action clause.

A purely conditional CARE statement is an inquiry. In response to such an inquiry, the computer will provide the list of patient numbers for which the conditional statement was true. Coupled with the compact summary and a statistical analyzer to be described later, we have a system for supporting legislative decisions in the clinical realm. What is the best treatment for hypertension (high blood pressure)? We can examine our experience to see. How valuable is the serum calcium as a screening test? We can look at the total number of serum calciums, the number that were abnormal and the consequent course of the patient. We can confirm intuitive impressions that a particular disease is too frequently escaping notice or that a particular therapeutic intervention is too often ineffective. We can do retrospective clinical research that has in the past required laborious hand perusal of thousands of pages of medical records.

In summary, the inquiry capability in concert with its two support modules gives us access to the distributed wisdom otherwise locked up in a large number of individual medical records.

CARE statements containing a conditional and an action clause have different functions than those containing only conditional clauses. We call them executive statements because they serve executive functions. A common clinical task is to execute rules of the form: If a given condition is detected, then initiate a given response. An examination of physicians' orders on any medical ward will reveal statements of the above form written to the nurse and attest to the fact that the physician need not be directly involved in all arcs from condition to response. For diabetics, we see orders written to adjust the insulin dosage according to the level of sugar in the urine. For heart patients, the physician writes to adjust the anti-arrhythmic medication according to the number of extra beats demonstrated on the cardiac monitor. For a large number of conditions, he will ask to be called if the blood pressure drops below a specified level. All of these orders are written by the practitioner to be executed by the nursing staff conditioned on the value of one or more parameters. These are called standing orders. We envision a role for the computer analogous but more extensive than that just described—though perhaps not quite as independent. In both inpatient and outpatient CFs, there are multitudinal tasks which can be described in terms of cookbook standing orders. These all involve a detection step and an action step. In general terms, there are many different classes of such tasks: the detection and response to a recent change in a parameter, the detection of a parameter abnormality which influences a drug's metabolism and the consequent appropriate dosage adjustment, the detection of adverse drug effects by regular checks on indicator parameters, and the initiation of the proper "work-up" of an isolated abnormality. We believe that such tasks constitute a significant burden on the primary care practitioner and are subject to error. By describing such tasks in terms of CARE statements, the burden can be transferred to the computer. It is clear that the computer (at the present) cannot physically accomplish the actions, thus the analogy with standing orders to a nursing service is not perfect. The computer can only produce a list of suggestions which must be executed by a human intermediary. We believe that at least initially the physician should be the intermediary and sieve the suggestions for appropriateness.

To give a feeling of the CARE language without going



```
          SURVEILLANCE REPORT          08-MAR-74

[redacted]


CONSIDER MEASURING:
   URIC [ 31-OCT-72] TO MONITOR : DYAZIDE

   DIAS BP [ 26-FEB-74] TO MONITOR : DYAZIDE


CAUTIONS:
   DIGITOXIN
        THIAZIDE DIURETIC : INCRFASED TOXICITY
        PHENOBARBITOL : DECREASED EFFECT
   DYAZIDE
        BUN= 24 => INCREASED RISK OF METAROLIC TOXICITY
           MIGHT REQUIRF CHANGING TREATMENT REGIMEN
        URIC= 9.8  & K+= 5.9  => CAUSE OF METABOLIC TOXICITY
           MIGHT REQUIRE D/C'ING
   FE SO4
        ANTACIDS : DECREASED EFFECT



  DO NOT PLACE IN CHART



REFERENCES AVAILABLE THE TWO KEY REFERENCES:
      A PRACTICAL QUIDE TO DRUG USAGE...  IN RENAL FAILURE
        BENNET ET ALL.     JAMA 214:1468-1475
      DRUG INTERACTIONS
        HANSTEN, LEA & FEBIGER , PHIL. 1973
```

Figure 3

into great deal of detail we present a few examples. Inadequately treated high blood pressure could be addressed by the following CARE statement:

> If "dias BP">100 then if last "BP meds">0 then increase "BP meds" because of under-treatment, else start "BP meds" because of absent treatment.

To declare that a parameter is important conditionally we write the following:

> If "diabets meds" then order "urine glucose"

The application of these rules to a CF is accomplished as follows: The practitioner develops the set of CARE rules which reflect the needs of his own practice. These are used to analyze his patients in a process we call surveillance. Surveillance can be initiated by a patient visit or at regular intervals by the passage of time. At each surveillance the patient's record is tested against the entire set of CARE statements. For a given patient, if the conditions of an executive statement are satisfied, two types of information are saved: (1) the time and value of the specific parameters which are satisfied; and (2) the consequent action clause. Two reports reflect this information, the encounter form and the surveillance report. Test suggestions appear in the "orders" area of the encounter form to facilitate the ordering of that test as shown in Figure 1. Conditionally important parameters whose conditions are met appear in the "observations" section to facilitate their entry into the system.

The surveillance report is best explained by example. Figure 3 shows such a report based on a limited set of CARE statements which relate to drug usage. Drug monitors are suggested at the top of the report, drug interaction and test results with important therapeutic consequences appear under the caution heading. One might notice from the flow sheet for this patient (Figure 2) that the recommendation to discontinue Dyazide was heeded.

In addition to the support of executive decisions, the system assists the actual prescription writing process. The conventional prescribing function requires that the patient's name and address and the prescribing instructions be written once on an individual slip for each active medication. With the encounter form, only new or changed medications require writing. For maintenance medications to be continued, the physician marks a single bubble and initializes the preprinted prescription. Only drugs under the controlled substance act such as narcotics require a separate prescription. This exception causes little inconvenience since they are rarely prescribed in an ambulatory setting.

Throughout the discussion of the CARE level we have not mentioned diagnoses. Presently, we do not store them. Active development is proceeding at an international level on a diagnoses and complaint code for ambulatory care. We would prefer to wait until that solidifies before we commit ourselves to a large dictionary of diagnoses. Our plan is to present the active diagnoses as a list along the far right-hand column of bubbles in the bottom section of the encounter form.

Operational experience

The CIS has been in operation at the clinical level since July 1, 1973. Since then we have accumulated records on more than 2000 patients in four different clinics, the diabetes, the renal, one session of the general medicine, and the nurse clinician clinic, and have been providing the three previously mentioned reports for all patient visits to the above mentioned clinics. More than 200 different clinicians at varying professional levels—senior medical students, interns, residents, staff physicians and nurse clinicians—have been exposed to these reports in one or more clinics. These care providers have been cooperative and have complied to the rigid requirements for inscribing block print numbers on the input forms. In the diabetes clinic, the character error rate was 1.5% of which half were automatically detected by the optical reader.

The time saved in writing prescriptions is spoken of by all the practitioners as being ample justification for use of the system, though the temporal savings have not been documented. In one study of eighty patient visits to a single staff physician, his average time per patient was 23 minutes with the conventional record and 16 minutes with the computer record. The summary report is appreciated but only after the patient has accumulated a computer record of some substance. The surveillance report has been used in the diabetes clinic since November of 1973 where it produced a measurable difference in physician response to abnormal lab results and the ordering of tests to monitor drug effectiveness. The acceptance of the physician in the diabetes clinic is remarkable because most of the care providers rotate on a monthly or bi-monthly basis. We had not expected the transition from standard record to the computerized version to be so smooth.

Ancillary service modules

As was mentioned previously, the most efficient method of trapping data which flows from the ancillary services to the clinical level is to computerize the ancillary services. This is our goal. At present we have two such systems in operation.

The lab system has been reporting results to the ward since November of 1974. Information processing in the lab consists of a two cycle process. Test requests arrive at the lab grouped by patient. The requests must be re-organized to conform to the organization of the lab, that is, grouped by test station, each of which performs a single battery of tests. Common to every lab is the fact that an individual worksheet is produced for each work station. The worksheet is simply a list of patients whose samples are to be analyzed at a given work station. The analysis is done; the results are recorded on the worksheet, transferred to the request form (i.e., re-sorted by patient), and distributed to their originators. In our system, test requests are logged via a terminal. The worksheet is a turnaround document produced by the computer on which results are inscribed as block-print numerics for direct optical reading into the computer. Both the initial and final sort is a computer rather than human sort. Multiple reports for multiple purposes (blood drawing lists, label, master logs, physician reports, ward reports, etc.) can be produced at will by specifying a command string. The system is flexible, robust and simple.

At the present, all tests performed by the chemistry, hematology and portions of serology section are handled by the computerized system. Test results for patients whose records are maintained by CIS are saved for batch updates to patient records.

The pharmacy module has been operational on a pilot basis since October 1974. A patient within the CIS presents his encounter form which bears his prescriptions to the pharmacy. After the necessary identifying information for the patient has been entered at the terminal, the computer cycles through all the active drug orders for the patient. These orders match the prescriptions printed on the encounter form, and the pharmacy clerk simply discontinues, repeats or changes the orders as the physician has indicated. New prescriptions are entered after all the active orders have been processed. For each prescription, the computer translates the prescription as written by the doctor (the sig.) into instructions to the patient, selects the tablet or bottle size from the pharmacy inventory which best fits the specified dosage, calculates the amount of medication to dispense, and prints a label for the medication bottle. Additional features aid the pharmacy in controlling its inventory.

SYSTEM IMPLEMENTATION

Since space does not permit a total description of the implementation of CIS, we will limit our description to some of the distinguishing features.

Data base management system (DBMS)

With the exception of the patient's medical record file and a few minor files, all data storage and retrieval is driven by a table of file descriptions and performed by a single I/O function and a set of encode/decode functions. The data base is divided into a number of logical files which may be contained in one or more physical files. All records contained in a single logical file are of the same size and format. A record may be between 2 and 512 bytes in length and can be divided into as many as 128 fields. Fields are of 9 types, scaled integer (2 bytes), floating point, variable sized scaled integer (from 1 to 7 bytes), date, time, pointer, string, packed string (only alpha-numerics) and bit string. Application programs access fields symbolically. Using the stored file descriptions, the I/O function determines the sector address, byte offset and length of the data requested. The data type is checked and the field is decoded if necessary. Records may be related in tree structured hierarchies (i.e. tree entry scheme).[2] The DBMS maintains all pointers for the application programs. From any record tree, a program can access fields in that record or fields in any record in the subtree below it. The I/O function performs error checking and provides some security in that only authorized programs are allowed to change the contents of a given logical file. For direct addressing of records by content, a hash table is created. A logical file is required for each hashed field.

We have a set of utility programs for storing, editing, listing, sorting, rebuilding and packing the files maintained by the DBMS. A command language is used to specify the action to be performed, the path to be taken through the tree structure, and the fields to be accessed. The fields are specified by logical identifiers, and if we had the core space, could be represented by text names. Unfortunately, with our limitations numeric "names" must do. The application programmer does not enjoy the luxury of a command string. He must pass a variable length integer array to the I/O function.

The advantages of this DBMS are twofold. First, file maintenance is simplified since common utility programs are used for all of the different DBMS files. And second, program code is decoupled from file structure. The time overhead to the system is insignificant. We are I/O bound and the DBMS is efficient in its use of I/O. However, there is a significant core space overhead required by the I/O function and the core resident file descriptions.

The statistical module

Since tallies, counts and averages are important to the legislative process at every level, we have developed a statistical processor for our data bases. Using a command string analogous to that presented above, one chooses the fields of interest from his logical file. Three different statistical outputs can be produced for each of the specified fields: (1) averages, (2) standard deviations, and (3) histograms. Histograms are defined over a partition of either equal or unequal spacing. A set of cut points is entered to define the desired partition. Averages and standard deviations are only defined for numeric fields. The histogram output can be generated for any field type, including strings. Thus, we can create histograms of the age of our patient population tallied over ten year intervals, or that of the alphabetic distribution of names tallied over the letters of the alphabet.

Many of our statistical questions are of a conditional nature. For instance, one may not want the average blood pressure of the population, but the average blood pressure by patient age. The statistics module can produce such information. For conditional statistics, the user must specify conditional fields as well as target fields. A partition defined by cut points must be specified for each conditional field. By sorting the records of interest by the conditional fields, the task of generating conditional statistics is converted into a simple repeated application of the process defined above.

The statistical module can be used by all files in the CIS. For administration, it can provide work volume statistics by clinic, by parameter, by patient type, by time, etc. For the clinical lab, it can provide quality control statistics. For the clinician, it can provide drug usages statistics and the incidence of test abnormalities.

Hardware and software

The computer which supports the CIS is a Digital Equipment Corporation PDP 11/45 located in the Regenstrief Institute at Marion County General Hospital. The PDP 11/45 is a midi computer with a word length of 16 bits and a cycle time of 300 nanoseconds. It operates as a time shared system available seven days a week, twenty-four hours a day except for periodic maintenance. Routine time sharing operations have been carried out on our system for more than two years. The central processor has 80K words of core memory, and 3.6 million words of disk memory. By the time this report is published, we will have acquired a 43 million word mass storage disk with expansion capability to 344 million words. The operating system, supplied by the vendor, is called RSTS/E. It provides a multi-user, interpretive environment for programming in BASIC-PLUS, which is a very powerful extension of Dartmouth BASIC. It includes full string processing, flexible file handling, matrix instructions, Algol-like statements such as IF-THEN-ELSE, WHILE and UNTIL loops, and recursive subroutines. This language was designed as an application's language and has shown itself to be very suitable to our medical applications.

CONCLUSION

In summary, we are in the process of developing a CIS to serve all levels of ambulatory care, and the ancillary service and administration levels of inpatient care. In contrast to other such developments, we rely on computer printed and optically read turn-around documents rather than keyboard terminals for data input. We feel that a CIS can be much more than a passive repository of data, that it can be actively involved in executive decisions, particularly at the clinical level. The CARE language was developed for that reason. The use of a computer from a family of minis and midis provides great market flexibility for our system. At the lower end of the DEC PDP 11 line are systems which could be afforded by small group practices, and at the upper end, we believe, are systems which will be able to support large outpatient clinics such as our own. We are happy with our choice of application language. The overhead of an interpretive language is no drawback in an I/O bound system such as ours; the increased programming efficiency more than makes up for the short-comings of BASIC-PLUS.

We have not yet reached our goals. The data base must yet be extended to include diagnoses and narrative descriptions, the latter of which we will store in a tree structured code. Ancillary service modules are still to be developed for radiology, EKG, biopsy reports and nuclear medicine. Our patient population must be enlarged tenfold to encompass our clinical target population. We acknowledge that our goals are ambitious for a midi computer and an interpretive language. However, our experience to date suggests that they are within reach.

REFERENCES

1. Grossman, J. H., et al., "An Automated Medical Record System," *JAMA*, 224, pp. 1616-1621, June 1973.
2. CODASYL Systems Committee, *A Feature Analysis of Generalized Data Base Management Systems,* May 1971. Available from ACM.

An on-line centralized computer-coupled automated laboratory information system using touch-tone card dialer telephone and audio-response technology for test order entry and result retrieval*

by ARTHUR E. RAPPOPORT, WILLIAM D. GENNARO and
ROBERT E. BERQUIST
The Youngstown Hospital Association
Youngstown, Ohio

INTRODUCTION

An important element in the evolution of the American Health Care System is the emerging recognition that many patients currently being hospitalized can be treated adequately on an ambulatory basis, thus relieving the pressure for hospital beds, reducing cost and eliminating the patients' separation from the home and family. Ambulatory Care Facilities (ACF) are being developed for these purposes in many medical centers.

The diagnostic departments play an important role in such ventures and the clinical laboratory stands in the forefront of that effort. Successful laboratory participation in an efficient ACF depends on the ability to perform rapidly a wide spectrum of tests with maximum reliability, accuracy and precision and to transmit promptly the results of those tests to the ACF physician to permit him to prescribe appropriate treatment without undue delay. In order to achieve maximum public acceptance of the ACF concept, it is necessary that the ambulant patient not be required to wait an inordinate period of time for examination by the physician, procurement and testing of specimens, diagnosis and definitive treatment. The crowded, chaotic, Out-Patient Department of yesteryear is no longer necessary or acceptable.[1]

The Youngstown Hospital Association (YHA) Laboratory Information System (LIS) has been evolving over the past 20 years in fortuitous anticipation of these developments and suggests appropriate solutions to many of the problems which are perceived in ambulatory medicine.

The YHA comprises three, geographically separate, hospital units; North (540 beds), South (450 beds), and Tod Babies and Children's (TBC) (75 beds). The North and South Units are five miles apart and TBC is about 1000 yards from the North Unit. North and South possess conventional Emergency Departments which require urgently performed laboratory tests. TBC and the South Units have large, structured ACF's treating over 100,000 patients per year. Upon completion of a current construction program at the South Unit including new nursing stations and an ACF, a marked increase in ambulatory care is anticipated. In order appropriately to respond to the swelling tide of in- and out-patient tests, the Department of Laboratories embarked a decade ago on a program to restructure its facilities and operating modes, and is now capable of mastering tomorrow's challenges.[2]

LABORATORY INFORMATION SYSTEM (LIS)

The LIS consists of six major sub-systems as follows:

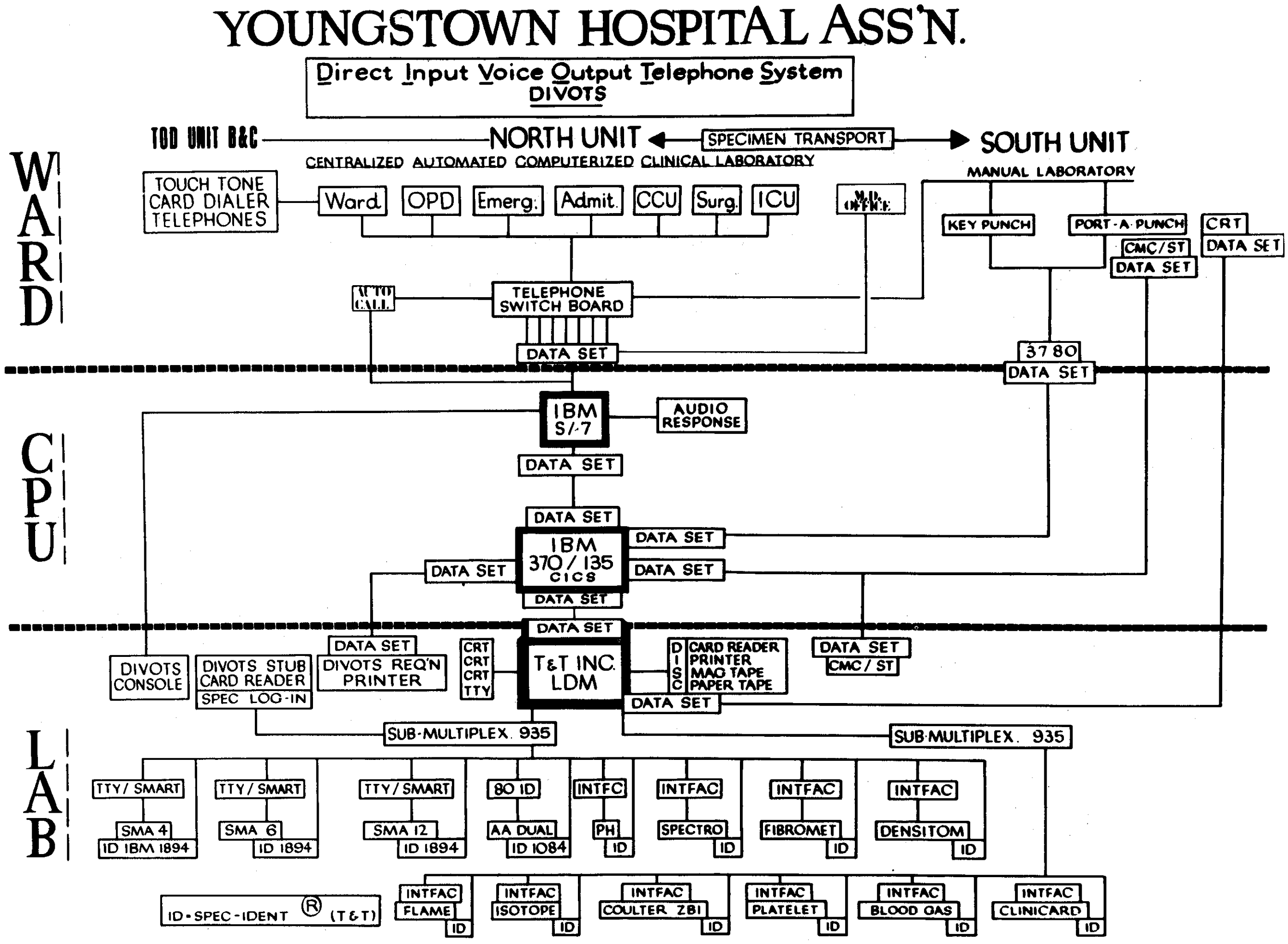
Centralization (Specimen collection and transport)
Examination (Test performance)
Documentation (Computerization)
Communication (DIVOTS and terminals)
Retention (Data storage and retrieval)
Administration (Staffing, cost control, records, etc.)

Centralization[3,4]

The Centralized Laboratory of the total hospital system is located in the North Unit and contains optimally sized staff and space, and sufficient equipment to perform most of the routine and special procedures required by all patients in all units of the YHA. Blood Bank donor processing and Serology testing are localized at the South Unit.

The South Unit acts as a satellite but possesses sufficient in-house staff at all times and facilities to perform all urgently required tests. In addition there are several phlebotomists responsible for the collection of specimens. These are transported by hospital personnel in a frequent automobile shuttle to the North Unit Laboratory Triage area when they are merged with specimens collected from

* This project was supported by Grant No. HS 06-05 from the PHS Health Resources Administration and Bureau of Health Services Research.



Flow Chart showing horizontal communication lines between the different departments of the various units. Vertical connections create a continuous, automatic, 2-way Laboratory Information System (LIS)

North and TBC patients. The specimens are immediately logged into the Spec-Ident (see below), centrifuged and brought to the appropriate laboratory division for analysis.

Without completely describing all details of this subsystem, it is self-evident that centralization allowing large volume testing leads to considerable economy of scale in terms of enhancing personnel productivity, space and equipment utilization. Thus it justifies substantial investments in the large scale, automated, analytic and computerized equipment to be described.

Examination

Laboratory tests are conventionally classified as Hematology, Biochemistry, Microbiology, Urinalysis, Blood Bank, Pathology and are carried out by registered Medical Technologists on specimens of blood, spinal fluid, urine, feces, and other human material under the supervision of clinical scientists and pathologists.

Tests are performed in three major modes: Automated, semi-automated, and manual ("handraulic"). In Figure 1, SMA 4, 6, 12 are automated and the devices, pH meter, Fibrometer, Spectrophotometer, etc., are semi-automated. These electronic instruments are coupled on-line to the in-laboratory dedicated computer by appropriate interfaces (SMART, INTFC) and multi-plexors.

The computer (LDM-Laboratory Data Manager), manufactured by T & T Technology, Inc. (Madison, Wisconsin), is built around the Data General Nova 1200 Mini-computer.

Automatic, machine readable specimen identification is achieved by ID (Spec-Ident), punched stub-card readers (below) which also communicate directly with the LDM. The test instruments emit a wide variety of digital, wave form or steady-state analogue, linear and non-linear signals and the data are reduced to final concentration either by hardware or software in the LDM.

Results of manual tests such as Urinalysis, Blood typing and Bacteriology, are entered into the LDM by laboratory personnel through the use of Port-A-Punch cards or terminal keyboards, or CMC/ST.[8]

The automated and semi-automated devices permit complete hands-off performance, eliminating manual transcription of collection, loading or work lists, result calculation and specimen identification, all functions carried out by the computer.

Documentation—Computerization[5,6,7]

The YHA centralized computer unit (CPU) consists of an IBM 370/135 System. Using CICS as a teleprocessing monitor, message switching is accomplished through terminals and the other computers. All patient demographic, medical and fiscal information is on file (PMF).

The IBM S/7 is a 20K mini-computer with a disk module, audio-response unit and possessing teleprocessing capability. One disk pack contains an 852 word audio-vocabulary of medical and laboratory administrative words. Figure 1 also demonstrates the configuration of the telephone network between the various units, the LDM, the 370 and the S/7 computers and their terminals.

The paperwork flow of a clinical laboratory may be divided conveniently into three major sectors. Front End, Middle and Rear End.

Front End

Front End Documentation includes the creation of the patient's basic demographic data base within the CPU. This includes all of his salient personal data such as sex, age, address, clinical diagnosis, previous admission to the hospital, financial status, etc. These data are inputted on line into the CPU from Admitting or ACF through terminals at the time of admission. The PMF is identified with a unique hospital number possessing a self-check digit; a non-detachable wrist band is applied. This bears a printed label describing the patient's identity and small pressure labels bearing the hospital number (infra).

Order Entry Routine

Physicians request tests from the laboratory by writing their orders in the patient's medical record. The actual ordering process is accomplished by nursing ward personnel. At YHA we have created an audio-response system—DIVOTS (**D**irect **I**nput **V**oice **O**utput **T**elephone **S**ystem) which permits the generation and transmission of those orders directly from ward or ACF to the laboratory by telephone-computer coupled technology.

DIVOTS requires the use of Touch-Tone (T.T.), card-dialer telephones located at the nursing stations (Figure 1). Currently there are three trunk lines from the hospital switchboard connected by WE403E modems to the System/7. It also possesses a commercial line bypassing the hospital switchboard to permit physicians to use DIVOTS from their office or home or to insure privacy.

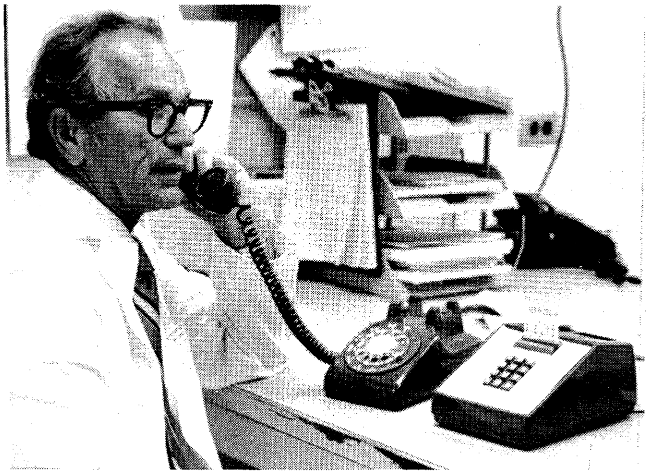


Figure 2—Photograph demonstrating the Touch-One, Card-Dialer pad attached to a conventional rotary telephone. Note the insertion of the patients' Dial-A-Card in the reader. The physician is either ordering tests or listening to laboratory results.

YHA is in a rotary telephone region and thus it is necessary that a T.T. pad, card-dialer accessory be attached to all rotary phones on Nursing Stations, ACF and special treatment areas including CCU, ICU (Coronary Care and Intensive Care Units), Surgery, Dialysis and Emergency Departments (Figure 2).

Physicians' offices have the option of using T.T. pads or a portable attachment which can convert a conventional phone to T.T. capability (Interface Technology, Inc.). This consists of a transmitter placed over the mouth piece and is attached to a battery-powered T.T. pad. Thus, any telephone anywhere can be utilized efficiently, economically and rapidly.

To order a test, the DIVOTS extension number is dialed on the rotary phone. System/7 responds with a vocal message to enter a specified request. A unique, previously prepared, patient's dialer-card permits automatic entry of his hospital number into the system which also may be done manually without a card. Within three seconds, the audio-response system verifies the identity of the patient by spelling out the first six letters of his name. Other data including the patient's ward and the time of the physician's order, are entered followed by test identification numbers as listed in an available Directory.

As soon as the test code is entered, DIVOTS states its name, thus verifying the request and assuring absolute accuracy of the two most vital test reliability factors, the name of the patient and the name of the test. If incorrect numbers are entered, the word "Error" is heard. Special instructions may be added through appropriate code numbers. As soon as the transaction is completed, a printer located in the laboratory and connected to the IBM 370, may print that patient's demographic, logistic and test information on appropriately designed, sequentially numbered and prepunched stub-card requisitions (Figure 3).

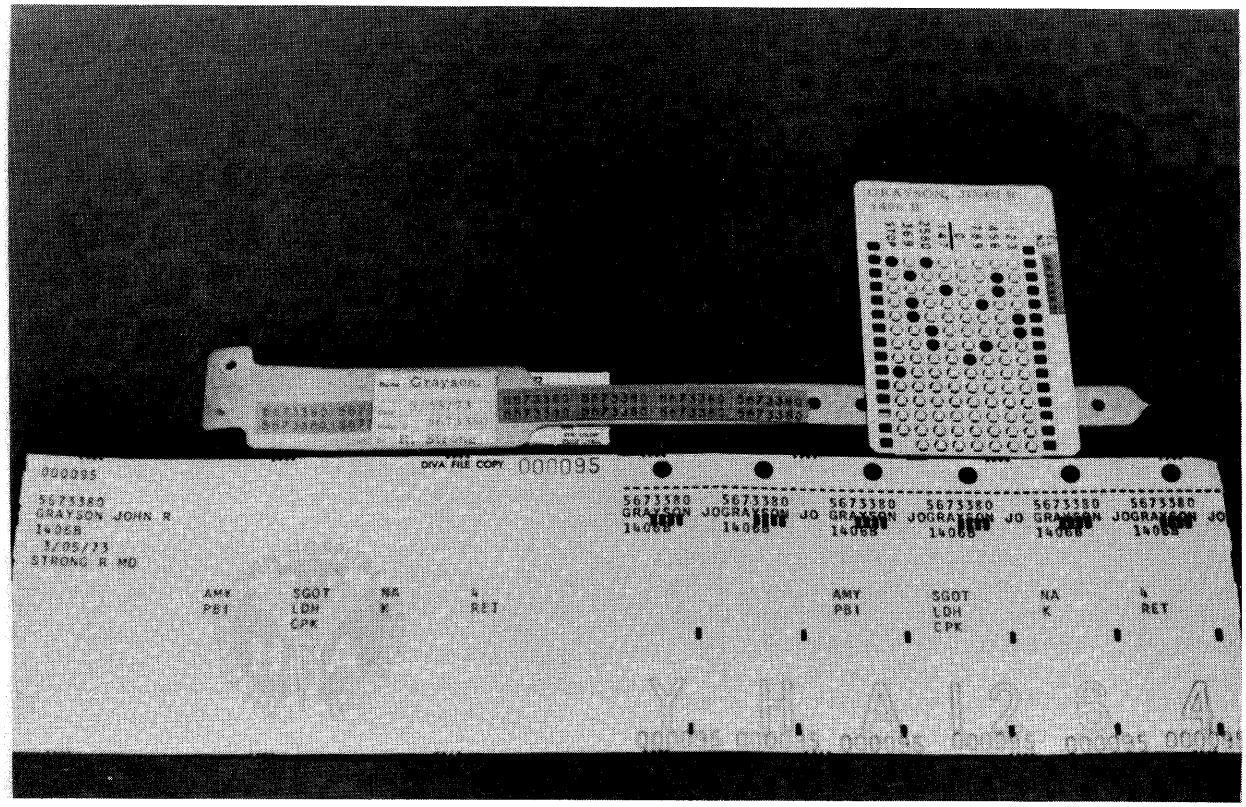


Figure 3—This displays a patient's wrist band, his Dial-A-Card and a completed DIVOTS requisition. The wrist band possesses human readable, patient identification data and bears pressure labels imprinted with the patient's hospital number. The Dial-A-Card has been punched with the same number.
The left half of the requisition is the Audit file copy. The right half possesses six, pre-punched individual, stubs each bearing the patient's hospital number, name, location and tests requested.

The patient's hospital number is automatically collated with the preprinted and punched requisition bearing the specimen number. This is accomplished as follows:

The number of the first sequentially printed and punched, fan-folded requisition is initially entered into the 370 manually, through the LDM. The 370 automatically merges the inputted requisition number with the patient's hospital number and thus achieves identification of each requisition according to the patient and his data base. This merge is continuously carried out and monitored automatically by the 370 on all subsequent patients' orders and requisitions.

This process resembles the conventional method employed in industrial organizations where the employee's number and pre-numbered check are collated. The 370 simultaneously creates a record of this transaction which is sent to the LDM and to the billing files within the 370 for further processing. Thus, the patient's identity may be ascertained from the specimen number in the LDM. The clock times of the physician's order and DIVOTS input are automatically noted and the elapsed time is calculated.

Specimen collection and identification

Laboratory personnel removes the pre-printed and prepunched requisitions from the printer. The requisition comprises an audit copy and six perforated stub-cards, each bearing the specimen number and the computer generated patient demographic information. The technologist goes to the patient and obtains the blood specimen. The printed identification data included on the patient's wrist-band are rechecked visually against the requisition. One of the pressure labels bearing the patient's hospital number previously attached to the wrist band, is removed and attached to the stub card permitting comparison with the printed hospital number and thus verifying the patient's identification.

Various types of blood specimens in different tubes are collected. Stubs are attached to those tubes by means of special rubber bands.

Upon return to the laboratory Triage area, the stub-card is inserted into the Spec-Ident which identifies and records the time of arrival. The specimen, with attached stub-card, is then centrifuged if plasma or serum is required or remains as whole blood, i.e., for Hematology tests. The specimens are then distributed by Triage personnel to the testing divisions and the actual bench-side examination is carried out.

In the automated and semi-automated, on-line systems, both specimen number and test data are merged and filed within the LDM automatically. Where manual testing is performed, the results are written on cards by the technologists and these data are key-punched or the technologists enter results directly on Port-A-Punch cards. All cards are read into the LDM and transmitted to the 370.

Middle documentation

Middle documentation may be roughly defined as that computer process which performs A/D conversion, peak picking, peak holding and similar types of data reduction. Through appropriate programs in the LDM, the instrumental outputs are compared with values of primary, reference standards and other materials, and computation of the patient's final results is carried out. Numerous quality control data checks are performed simultaneously to establish the accuracy and precision of the results and to create permanent records for validation. These results are collated with the specimen numbers and these data are merged with the patient's hospital number in the PMF in the 370.

Rear end documentation

After the test data are entered into the PMF, they become available for transmittal to the physican.

Interim results are printed at 12:00 noon daily on a Ward Report which is sent to all stations within the institution. The billing system is updated, exceptions are noted, and quality control reports are generated for permanent file.

These printed reports are transmitted by messenger to the appropriate nursing stations.

Communication

Audio-response communication of results by DIVOTS can be achieved. As soon as the test is filed in the patient's PMF, DIVOTS may procure the results and transmit them audibly to the inquiring physician as follows. DIVOTS is telephoned as described above, a different transaction code is employed and the patient's hospital numbers are entered. The physician's private identification number is keyed in to permit only him to obtain the confidential medical information. He enters the test code and DIVOTS spells the patient's name, states the name of the test, the day the test was performed and the test results including the degree of normality. Physicians in the ACF, their offices or homes or, by long distance lines from distant locations, may interrogate the PMF at any time if knowledge of the patient's and test numbers is available.

Autocall

An important extension of DIVOTS rear end communication ability is the Autocall System which is used for:

- Certain divisions of the hospital with a "high need-to-know" factor such as CCU, ICU, Surgical Recovery, Emergency Room.
- Certain requests such as Emergency, ("stat"), requests.
- Specified substances or procedures such as Glucose, Potassium, drugs, Prothrombin time, etc.

These have been identified in the 370.

As soon as test results in any of these categories are transmitted by the LDM to the 370, the Autocall subprogram is initiated. Since the patient's ward and its telephone number is known in the PMF, the 370 informs the System/7 to dial the telephone at that place. Upon answering the phone, the hearer hears a series of chimes from the audio-response disk, identifying it as a DIVOTS call. The hearer indicates recognition of DIVOTS by entering the ward code. DIVOTS then automatically transmits the patient's name, the test name and the result repeatedly until the hearer hangs up.

Autocall therefore represents a sophisticated yet economical method of achieving automatic, rapid, and reliable data transmission to physicians requiring that information in situations where those data are of the highest medical importance. "Don't call us—we'll call you."

Although DIVOTS represents a widespread audio-communication system, there is also a network of CRT's in various places within the laboratory for test process control, data base maintenance and test result entry. Terminals are also available in the South Unit for reading punch cards and printing patients' results on the Ward and Patient Summary Reports simultaneously with similar activities at the North Unit (Figure 4).

Free narrative test reports such as Pathology diagnoses and consultations as well as any other free, unstructured reports and discussions, are entered into the PMF by means of the Communicating Magnetic Card/Selectric Typewriter (CMC/ST—IBM). These inputs are usually generated as byproducts of the transcription of the usual conventional pathology, EKG and microbiology reports, and thus represent a low cost, extremely efficient method of entering edited input. The magnetic card data are transmitted to the 370 by Data Sets through a dedicated dial-up port, shared with other CMC's in the institution.[8]

A special printout of laboratory data for laboratory operations, permanent quality control records and statistical data, are generated by the LDM Printer.

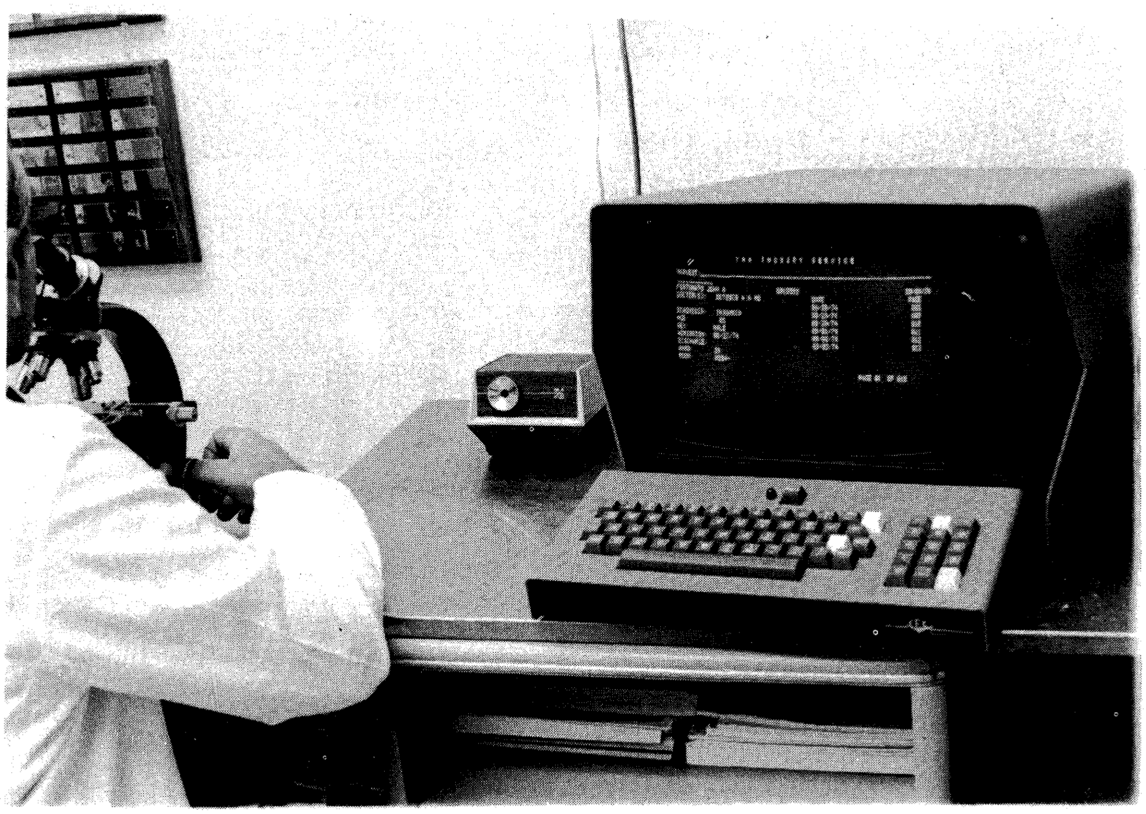


Figure 4—Note the availability of a CRT at the pathologist's microscope work-bench allowing him to retrieve the total PMF and Quality Control data thus allowing correlation between the anatomic findings with clinical pathology data.

Retention (data storage)

All current laboratory information is stored in the 370 on disk and is immediately retrievable by DIVOTS, CRT's or printers. Each patient's laboratory record is retained during his entire period of hospitalization plus 15 days after departure from the institution at which time the stored data is transferred to magnetic tapes for permanent files.

The PMF may be interrogated at anytime. There are no paper files.

Administration

All Sub-systems exert an important impact upon the administration of the laboratory and thus, on the quality and cost of patient care, by increasing economy of operation and optimizing the productivity of the staff.[9] Other benefits are:

- Reduction of manual transcription of test requests by nursing personnel.
- Elimination of time and expense in transporting requests from ACF and nursing stations of the laboratory.
- Elimination of expensive forms.
- Elimination of need for patient and test data input by keypunching or other terminals.
- Reduction of elapsed time between initiation of request by physician and availability of laboratory results.
- Elimination of all manual filing of millions of individual paper reports by storage of these data on magnetic tape.
- Creation of charges or credits for services automatically accomplished when the test request is initiated and result filed.
- Generation of important statistical and other administrative data as to the volume and types of procedures carried out.
- Documentation immediately of quality control of test.
- Achievement of maximum reliability of test data, patient and specimen identification.
- Availability of systems for backup support in the event of technical instrument or computer failure.
- Improving and accelerating the physician decision making process by furnishing him error-free data by phone wherever he may be.
- Creating printed documentation of the reliability of all steps of the test path pursuant to ethical, professional and legal requirements of quality care,

and according to the Rules and Regulations promulgated by Federal and State Laws (Medicare, Medicaid, Interstate Improvement Act of 1967—CDC, HEW, Social Security) or institutions.

The advent of Professional Standards Review Organization (PSRO), the Inspection and Accreditation Program of the College of American Pathologists, and similarly organized programs dedicated to the improvement of laboratory performance imposes mandatory requirements to create and retain extensive records during the test performance. These are automatically accomplished by appropriate software. In our completely automated systems, these requirements do not represent a significant added manpower, financial or systems burden.

CONCLUSION

From the foregoing, it should be obvious that the LIS and the Sub-systems which have been developed at The Youngstown Hospital Association are extremely useful in maximizing the functions of an ACF. We have described the need for, and means of, achieving rapid, error-free, patient and sample identification, procurement and transportation to the centralized laboratory, rapid automatic performance of the test and prompt transmission of the test data to the attending physician in the ACF while his patient is still there and can benefit by appropriately guided treatment. Thus, we believe that the major professional impact of the Youngstown LIS is focused on the "physician decision-making process".

Upon early receipt of the laboratory, EKG and other diagnostic information, his diagnosis may be confirmed, strengthened, changed or expanded and his treatment may be fine-tuned. Additional tests, other diagnostic (X-ray, EKG) procedures may be requested. The type, amount and form of medication may be reviewed, modified or discontinued. Consultation by other physicians may be requested. Admission to the hospital may be evaluated. All of the subtle decisions which underly professional care are buttressed by prompt availability of laboratory data.

The fact that the turnaround time is so short implies that these decisions can lead to performance of any of these additional acts while the patient is still in the ACF. This reduces the number of visits and eliminates the need to return at later dates.

Further studies are being carried out to determine the cost effectiveness, the systematic advantages and the technical requirements of the present system with the view of expanding the current LIS into a broader Medical Information System (MIS) by including such Sub-systems as Pharmacy, Surgical Scheduling, Dietary and Patient Logistics.

Industrial Engineering and Systems Design studies of this project are currently being carried out by members of the Department of Industrial Engineering of The Youngstown State University who, as Consultants, also act as our severest critics.

REFERENCES

1. Rappoport, Arthur E., "A Clinical Laboratory Designed for Limited Service," *Hospitals, J.A.H.A.*, Vol. 41, pp. 59-63, February, 1967.
2. Rappoport, Arthur E., James R. Hill and William D. Gennaro, "Simultaneous Performance of Twenty-Five Different Tests Per Minute in a Centralized Cybernetic Clinical Laboratory," *Cybernetics, Artificial Intelligence and Ecology*, Spartan Books, 1972.
3. Rappoport, Arthur E., Wilbur R. Taylor and Richard P. Gaulin, "What the Modern Laboratory Must Include and Where to Put It," *Modern Hospital*, November 1973.
4. Rappoport, Arthur E., "Laboratory Design;" Chapter 26, *Laboratory Medicine*, Harper & Row, 1973.
5. Rappoport, Arthur E., "Computers, Information Retrieval and Data Storage," *Laboratory Medicine*, Chapter 23, Harper & Row, 1973.
6. Rappoport, Arthur E., William D. Gennaro and Robert E. Berquist, "Two Decades of Evolution of Laboratory Information Systems at The Youngstown Hospital Association," *AJCP*, March, 1974.
7. Rappoport, Arthur E. and William D. Gennaro, "Instrument Series: Report #26, Clinical Laboratory Computer Systems-Part III," *Lab World*, December 1974.
8. Rappoport, Arthur E., Robert E. Berquist and William Gennaro, "The Communicating Magnetic Card as a Word Processor to Enter Pathology Narrative Reports and Clinical Laboratory Results into a Patient's Computer Record," *Laboratory Medicine*, in press.
9. Rappoport, Arthur E. and William D. Gennaro, "The Economics of Computer-Coupled Automation in the Clinical Chemistry Laboratory of The Youngstown Hospital Association," Chapter 10, *Computers in Biomedical Research*, Vol IV, 1974.

Architecture for a graduate level educational program in the area of computer systems in medicine

by LAURENS V. ACKERMAN
Rush Medical School
Chicago, Illinois

and

DANIEL K. HARRIS
American Medical Association
Chicago, Illinois

INTRODUCTION

A professional discipline, in its embryonic stages of development yields practitioners who, through interest and/or circumstance, avail themselves of fragmented knowledge from other disciplines, synthesize intellectual common denominators, and apply these common denominators consistently to a multiplicity of problem situations. Eventually, the embryonic discipline will generate additional knowledge from within, which is generally considered to be a necessary precursor to formal acknowledgment by society that a new discipline has in fact emerged.

The practitioners of a new discipline will, at first, be convenient instruments for use in sporadically occurring situations requiring the application of their specialized knowledge. Over time, however, sporadic needs may become commonplace—more so than the practitioners trained to satisfy them. It is at this juncture—the transition of a profession from a convenience to a necessity that the mettle of the profession to produce qualified practitioners is tested.

The evolution of computer expertise in the medical environment has tended to emerge according to the pattern suggested in the above scenario. The computer science and systems professionals serving the medical community represent a diversity of backgrounds, both educational as well as experiential. Due in part to this diversity, fundamental, qualitative norms of performance for medical computing professionals have not surfaced. Indeed, the sources of computer expertise to serve the medical community are difficult to identify because of the lack of educational patterns as well as educational institutions offering programs in medical computer science.

PROGRAM OBJECTIVES

The purpose of the educational program described is to provide a structured, coherent, educational framework of sufficient breadth to permit the development of professional expertise in one or more of the following health system areas.

(a) Clinical systems development
(b) Medical administrative systems development
(c) Medical computer science

As the central theme of the program is to train professionals in the many and increasing facets of medical computing, several corollary interests surface naturally. These additional interests include:

(a) Establish the basis for continued research and development of medical computing systems.
(b) Develop a reservoir of expertise sufficient to meet the medical computing needs of the ambulatory care environment in addition to those of the medical center environment.
(c) Provide a basis for the development of continuing education programs for physicians and laymen with interests in medical computing.
(d) Provide a dynamic model for medical schools and other educational institutions to consider as a starting point in the development of similar programs.
(e) Develop professionals capable of communicating at the medical, computational and administrative levels in the development and administration of health systems.

HISTORICAL CONSIDERATIONS

The architectural components of the program offered for consideration have been influenced by the educational dynamics present in the commercial and biomedical engineering computer environments over the last twenty years. Analysis of these two apparently dissimilar environments reveals several commonalities which have combined to cause the current pattern in medical computing education and development.

First, both the medical and business environments have tended to obtain computer expertise from apparently logical sources. In commerce, the accounting function served as the source of manpower and in medicine, biomedical engineering was selected (implicitly) to develop needed expertise. This "selection process" has resulted in a traditional de-emphasis of computer and associated disciplines as legitimate activities in their own right. Until recently, the computer or management science profession did not exist in the commercial environment. In the medical sector, the computer profession continues to be represented as an extension of the biomedical engineering function. As the use of the computer is increasingly extended in non-bioengineering areas, this orientation, at least in some quarters, is beginning to change.

Secondly, computer education has had an on-the-job orientation at a period in time when the technological emphasis has been in developing new, more sophisticated computer hardware systems. The result has been that applications oriented programmers and systems analysts have tended to proliferate the same systems and applications under changing and more complex hardware configurations. It is not surprising therefore, to be confronted with a widening gap between hardware capability and the capability of the programmer and analyst to utilize it effectively.

Third, because the orientation in both medical and commercial computing has been device and application oriented, the emphasis on the man/machine interface has been decidely oriented toward the machine. Analysis of the influence on behavioral mechanics of the environments in which computers operate has not been an integrated component of the educational experience of computer professionals. The trauma normally experienced when a "hospital information system" is implemented in what is generally acknowledged as a complex, interrelated human organization is not surprising. The experience in this regard suggests that instruction in the behavioral and organization dynamics of human organizations is at least as important to system success as instruction in computer programming and systems analysis.

Finally, and perhaps most significantly, physicians, computer programmers and analysts, and administrators have tended to insulate themselves within their respective roles. Communication among functional entities is hampered partially due to terminology differences and partially due to role isolation. Systems development tends therefore to be unilateral in thrust and execution. The hospital information system is in actuality, an administrative information system. Clinical laboratory systems tend to be driven by individual computers without multiplexing capability for information exchange.

PROGRAM ARCHITECTURE

The program offered for consideration is graduate in nature. It assumes a general knowledge of fundamental data processing concepts and requires a knowledge of computer programming. Basic characteristics of the program are as follows:

1. The program offers a Masters or a Doctoral degree.
2. Program duration will vary from two years to four years, depending upon the candidate's academic and experiential backgrounds, as well as the type of degree sought.
3. The program would be open to, and encourage the participation of, both physician and non-physician candidates.

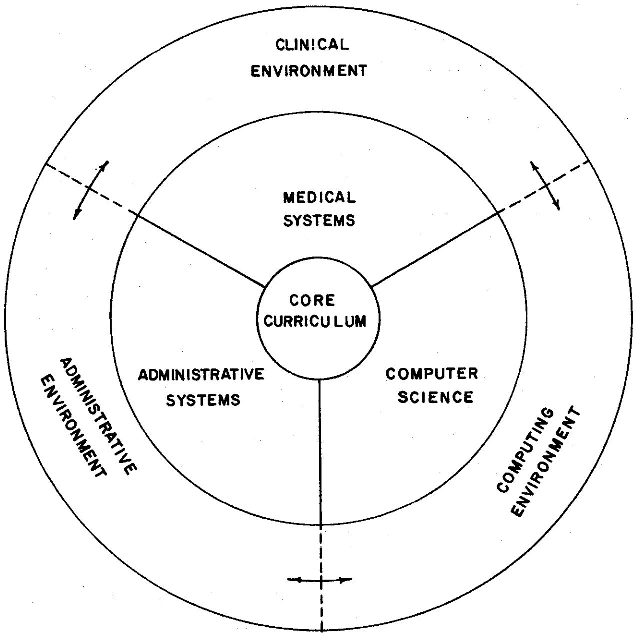
The program will offer a core series of courses in each of six major areas. These areas include:

1. Computer Science
2. Medicine
3. Mathematics
4. Administration
5. Systems Architecture
6. Computer Systems In Medicine

A key element of the program is the integration of each basic area with all others. Thus, the physician will have the fundamental knowledge necessary for effective communication with the administrator and computer scientist. The non-physician will be given essential medical knowledge necessary to communicate with the physician. Candidates in each area of interest, through the core

GRADUATE PROGRAM

OVERVIEW



courses, will be sensitized to the needs and terminology idioms prevalent in the other areas.

Course offerings are structured to: (a) provide insight into fundamental concepts, (b) establish the framework for development of design techniques and (c) provide an opportunity for the candidate to apply concepts and design techniques in a live environment.

Administrative component

The core courses in the administrative component of the program are intended to provide the candidate with detailed insights into the structure and operating components of the administrative sector of the health care delivery system. Core courses dealing with finance, economics and management activities as they relate to development of the health administration information base are presented. In addition to traditional administrative material, students will also be presented material concerning organization theory and analysis as well as the behavioral dynamics of group interaction in structured and unstructured environments. Thus, in addition to possessing the ability to define and design administratively related medical systems, students will be expected to have a clear understanding of the environment which these systems are intended to influence.

Systems component

Courses offered in the systems component of the program are intended to provide the student with:

(a) A fundamental appreciation of the concepts of systems theory and organization.
(b) An insight into the inter and intra structural mechanics of systems.
(c) An appreciation of systems techniques available for the development of clinical and administrative EDP information systems.
(d) Insights into the man/machine interface vis-a-vis medical EDP systems.

Students completing the courses in the systems component will be expected to be capable of applying systems development techniques consistently in a multiplicity of medical design environments as well as integrating discrete components of the medical environment into coordinated development activities.

Computers & medicine (administrative) component

This graduate program component provides the student an opportunity to intensively study medical administrative systems in various health care settings under varying conditions. Stress will be placed upon administrative medical systems whether in operation or in the process of being developed. Students will be given the opportunity to interact with systems designers and users. As system design requires optimization of available resources in the context of a given environment, students will be provided opportunities to design and program EDP systems stressing design options and quality assurance mechanisms both in a machine and man/machine context.

Medical component

Cognizant of the extreme complexity of medicine, the program does not emphasize the use of computers in systems analysis of medicine. Rather, the student is provided a concept of the system that he is analyzing. Although the system and computer person have many techniques which are theoretically capable of analyzing a system, the reality of systems analysis requires a delicate intertwining of theory and practice to produce an operational and practical system. Both the computer scientist and systems scientist, plus the administrator, must have a knowledge of the system in which they are functioning and in this instance, that system is medicine. An objective of this educational pattern is to provide an understanding of some of the problems encountered by physicians and other paramedical personnel, and solve these problems in terms of methodologies taught in the program. To this end, the student will be provided a basic medical physiology sequence in what might be called disease states, taken from a system perspective (as that is the orientation of the students) approaching a point where the student will attend grand rounds and understand the problems presented there. This represents a radical departure for most administrators and computer systems personnel who either feel caught up by the overwhelming difficulty of the medical sciences or feel more secure inside of their traditional roles. The medical component of the program, therefore, represents a unique opportunity for non-physicians to empathize and communicate effectively in the medical systems design environment.

Computer science component

Prior to entrance into the program, the student must demonstrate competence in a computer language as well as a basic understanding of fundamental computer technology concepts. Because of the balance between administration, systems, and medicine, with the computer as a unifying element, the student should gain knowledge of a business and a scientific language, probably COBOL and FORTRAN (with PL1, the extent depending on his interest). Additionally, there should be a comparative knowledge of other languages to permit the student to ascertain which language should be used in a particular application. Also, an important aspect of the student's education will be medical data base design, as this will be the basic building block for systems, computer and medical applications. Finally, it is desirous that the student have knowledge (at the assembly language level) of one multiuser, time share system so as to have an idea of the technical components in a large multiuser system employed in a hospital.

Computers & medicine (clinical) component

Inherent in the medical and administrative courses will be the various respective computer applications. However, in-depth courses in computers in clinical medicine are provided to aid the student in the development of design capability. The core section of this program is not intended to completely acquaint the student with the computer in clinical medicine, but it is thought very important to orient the student to some of the problems he is likely to encounter as well as possible alternate solutions. A course in clinical computer systems in medicine will specifically examine the computer as used in clinical activity. In pursuing this area the student will be required to develop a minimum level of competence in mathematics including matrix theory, statistics, and probability, in addition to demonstrating knowledge of current techniques in artificial intelligence and pattern recognition. Pertinent mathematical techniques addressing classification, clustering, automatic diagnosis, and decision techniques will be stressed along with current methods available to model medical knowledge. In addition to the theoretical level the student will also examine systems in use in university and nonuniversity hospitals and ambulatory settings in addition to examining various commercially available medical computer systems.

PROGRAM DESIGN

Because of the interdisciplinary nature of this program there is room for both M.D. and non-M.D. candidates to matriculate into a variety of operational, research and teaching roles. The program is designed for a multitude of people with different backgrounds and interests, offering a basic series of core courses enabling them to communicate at multidisciplinary levels and then follow various specialization tracks that emanate from the core. The M.D. candidates would not be required to take the physiology and disease system courses, but it is understood that the M.D.'s probably would require some remedial work thus equalizing the M.D.-non-M.D. course load. The tracks are then divided into three thrusts which would include administrative, clinical systems, and computer science. To provide students an experience in administrative and clinical live environments, internships are required of both types of students in both areas. Additionally there would be an activity called "Computer rounds," intended to attract other members of the hospital to provide a purposeful discussion. Inherent in this program is the concept that this is mainly an operational activity with provisions for students to gain a theoretical appreciation of concepts at work in the operational environment. The degree structure is organized to provide an "operational" and "theoretical" candidate with a similar core background. After approximately two years, the student would be given the choice of continuing his studies for a Ph.D. or elect to stop at a Masters and become active in the operational environment. The function of the Ph.D. is to educate a person to perform research in operational areas; whereas the Masters is established for an individual anxious to initiate activity in the health care environment without having to pursue the extended research of a Ph.D. activity.

SUMMARY

As was indicated earlier, the purpose of the educational program offered for consideration is to provide for a structured, coherent, educational framework of sufficient breadth to permit the development of professional expertise to influence advancement in the areas of clinical systems, medical administrative systems, and medical computer science. The root concept supporting the program structure is the emphasis of cross communication and training in the functional components of the health care delivery system. The computer, to be sure, does not represent an end point in itself. However, as the use of the computer proliferates in the health care community, the computer serves as a catalyst to draw heretofore isolated areas of the health care system into integrated focus, both conceptually and operationally.

There are several alternative methods available to train the expertise necessary to fill a growing technological need in the medical community. Traditional "on-the-job" training programs, such as those often employed in the commercial environment offer apparently fast solutions to the imbalance between supply and demand of medically oriented computer professionals. However, for reasons discussed earlier, these programs tend to lack consistency and depth, over the long term.

The program architecture offered for consideration is intended to serve as a basic model for graduate programs in computer systems in medicine. As is the case with any model, it has built into it the type of flexibility necessary to accommodate a changing, dynamic environment. However, in so stressing the need for flexibility, it also recognizes the need for unanimity among health care functions traditionally isolated in operation. The coalescing point is the development of a rational, well integrated system. With the delivery of health care as the principal focus . . . not administration, nor computation, nor theoretical medicine functioning as independent activities, but rather these important activities functioning in concert, as a whole.

Area Director:
William P. Stritzler
AT&T
Morristown, New Jersey

Day on banking—Today's challenge

Four two-hour sessions conducted by industry specialists and designed for middle managers of technical functions. An opportunity to gain insights into the emerging, exciting arena of Bank Data Processing.

Sessions will cover the following: (1) Electronic Funds Transfer Systems—a panel of knowledgeable members of the financial community discussing E.F.T.S. issues and answers. Perhaps the most widely talked about and least understood major business and technological activity in the financial community today, E.F.T.S. transcends Banking and should be of interest to all industries. (2) Implications of communication based systems technology on banking. A panel identifying opportunities, problems and solutions in this important area of Bank Data Processing. This area may also be of interest to non-bankers, as broad technical issues will be discussed as well as their particular application to banking problems. (3) Banking back office paper problems and approaches to solutions. A panel discussing various manufacturer approaches to reducing costs in this highly labor intensive activity. Of interest to anyone with an interest in the solution of complex systems problems related to high volume labor intensive activities. (4) Data Base progress in banking today. A panel reviewing what has been accomplished and how. An opportunity to understand how a major industry is applying Data Base technology to help address its systems problems.

All in all, a Day on Banking is designed to be of interest to any technical manager wishing to understand how one major industry is applying Data Processing technology to the solution of important business problems.

Area Director:
Bertram Raphael
Stanford Research Institute
Menlo Park, California

Innovative applications of computer science

For many years much of the research frontier of computer science as represented, for example, by the activities of artificial intelligence laboratories, seemed preoccupied with esoteric mathematical studies (such as self organizing systems, algebraic theory of machines, or resolution theorem proving), "toy" systems (such as games, puzzles, children's blocks), or far-out science fiction goals (such as robots for space exploration). Now many of these same laboratories are applying the techniques they have developed in the past to important, short term, real world tasks—and uncovering significant new research problems in the process. By working in new interdisciplinary teams, the computer scientists and the applications specialists have begun to develop an evolving series of novel systems whose potential value to our society is tremendous.

The *Innovative Applications of Computer Science* technical area at the 1975 NCC will consist of four panel sessions, each aimed at discussing interesting computer-based innovations that can produce wide spread practical benefits within a few years. The first three sessions—Innovative Applications of Computers in Education, Medicine, and Automation—describe some dramatic computer-inspired changes that are occurring in those respective disciplines. The fourth session—Knowledge-Based Expert Systems—presents a major methodological viewpoint for much of this work, and some of its broader implications.

Innovative applications of computer science in medicine

CHAIRMAN—G. ANTHONY GORRY
Massachusetts Institute of Technology

OVERVIEW—G. Anthony Gorry

For a variety of economic, social and geographical reasons, there currently exists a maldistribution of medical expertise with respect to the needs of the population both in this country and perhaps more markedly abroad. The particular characteristics of the computer, its large memory and the industry of its computing engine, suggest the possibility of its use to provide *replicable* and *distributable* packages of expertise. Attempts to build such packages, however, until recently have been confined to rather sharply circumscribed medical problems. Success has been somewhat limited, and the net effect of this work has been small when compared to the problems which exist in the health care industry. Advances in computer science, and in particular in the domain of artificial intelligence, offer hope that some of the major impediments to success in this area can be eliminated. This session will be concerned with current work in the area of computer science applications in medicine. Particular attention will be paid to the identification of the cutting edge of computer science and artificial intelligence technology in medicine and to the major problems which as yet remain unresolved.

Summary of Comments—Harry E. Pople

One of the major difficulties to be overcome in dealing with real-world clinical problems is that of irrelevant data. The issue is not just one of erroneous reporting by patient or laboratory, although these are often unreliable sources of information. The problem of irrelevancy also arises in cases where all data is manifestly correct but is produced by two or more distinct causal mechanisms. In dealing with such cases, the skilled clinician focuses successively on various aspects of the problem, disregarding as he does so those findings that are irrelevant in each context.

What enables the specialist so to perceive problems in the presence of extraneous and misleading data has been perhaps the most significant and perplexing question confronted in our research. Our results suggest that there is an underlying logic to the process, which can be programmed, enabling computer-based medical diagnosis that approaches expert levels of performance.

Summary of Comments—Saul Amarel

Computer systems with capabilities for convenient acquisition and management of knowledge in a given domain, and with facilities for intelligently performing processes of interpretation, planning and theory formation, have the potential for significant impact on research and practice in Biomedicine. Work in artificial intelligence is now at a point where it can provide the basis for the design and implementation of such systems. Advances in computer networking are opening the way for collaborative developments of such systems and for their shared utilization by groups of investigators across the country. In the last four years, we have been pursuing work along these lines at the NIH-sponsored Rutgers Research Resource on Computers in Biomedicine. During the past year our Resource has been actively participating in the computer-based SUMEX-AIM national network on artificial intelligence in medicine. An outline of our research activities will be presented with emphasis on work in Medical Modeling and Decision Making and in Modeling of Belief Systems. We have already obtained significant results in the areas of glaucoma consultation and in the modeling of common sense interpretation of social episodes. Results and experiences in these areas will be outlined, and future prospects will be discussed.

Summary of Comments—Bruce G. Buchanan

Two computer programs that have been developed at Stanford University demonstrate different applications of computer science to medicine. Not only are the problem areas different but the methods of approach represent viable alternatives for many similar problems. The first program, named MYCIN (after the common suffix of many antimicrobial drugs), helps physicians diagnose infectious diseases and devise appropriate therapy for them. It is an artificial intelligence program in which a large amount of expert knowledge has been stored in such a way that (a) the program's performance mirrors that of an infectious disease consultant, (b) the program's reasoning can be explained either during or after an interactive consultation session, and (c) the rules in the knowledge base are easily understood and modified.

A second computer-based system, the MEDIPHOR System, has been developed for the prospective control and study of drug interactions in hospitalized patients. The system employs a large, well-documented computer-stored data base of drug interaction information and a series of interactive programs to provide automatic notification to pharmacists, nursing staff, and physicians when potentially interacting drug combinations are prescribed. In doing this, it utilizes information entered at a central inpatient pharmacy to monitor drug use, create patient medication profiles, and generate drug interactions reports.

Summary of Comments—David G. West

In the past the biologist considered the computer to be an expensive tool for the exclusive use of the computer scientist. With the advent of minicomputer, data communication, picture display technology, and the increased sophistication of the user and the computer scientist, computers are being used as integral parts of the experiment.

For example, two of the problems being investigated at the Biomedical Division of Lawrence Livermore Laboratory are: to quantify normal and abnormal chromosomes by analyzing the DNA content of chromosomes in a cell; and to automate cancer detection by using a method of high speed analysis and physical sorting of cells and chromosomes. To solve these problems we are utilizing two multidisciplinary approaches which include extensive use of the computer technology. In the first approach, scanning electronics collects picture data and the information for evaluation is extracted by interactive image analysis and statistics. The second approach uses the flow systems technology, analog electronics, computer automated data collection and curve fitting, and statistics to provide high speed on-line analysis. Our current results enable us to detect a change as small as 8×10^{-15} grams of DNA in a chromosome, and to analyze and physically sort 1,000 cells per second. (This work was performed under the auspices of the United States Atomic Energy Commission.)

An "intelligent" on-line assistant and tutor—NLS-SCHOLAR

by MARIO C. GRIGNETTI, CATHERINE HAUSMANN and LAURA GOULD
Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

INTRODUCTION

NLS-SCHOLAR is an experimental system that uses Artificial Intelligence techniques to teach computer-naive people how to use the powerful and complex editor of NLS.* This teaching is accomplished by presenting a sequence of lessons. During each lesson the student may interact with the system by asking and answering questions, performing tasks which are posed by the system, and performing tasks of his own choosing. Tasks are actually executed using our own implementation of NLS EDIT.** Those tasks which have been posed are evaluated by the system, and the student is given encouragement, advice, and assistance.

NLS-SCHOLAR has been designed with the belief that procedural knowledge is best learned 'by doing';[2–5] along with the BIP system[6] it is an example of a new kind of Computer Assisted Instruction (CAI) system that integrates systematic teaching with the capability for the user to try out what he learns on the very system he is learning about.

NLS-SCHOLAR is designed so that it can also be used as an on-line help system *outside* the tutorial environment, allowing users to ask questions arising in their actual work, with NLS-SCHOLAR being aware of what they are doing and answering accordingly. Thus the system can take the lead at first, and fade smoothly into the background as users become proficient. This capability of integrating on-line assistance and training is an extension to the traditional notion of CAI.

Although NLS-SCHOLAR is an almost entirely new system, it has been very heavily influenced by previous or concurrent work. First to be mentioned is SCHOLAR,*** of which our system preserves the flavor and interaction characteristics. In terms of its underlying philosophy and approach our system owes much, in an explicit sense, to Brown's SOPHIE system,[10,11] while implicitly, by virtue of being written and developed in INTERLISP, the system is impregnated with the ideas that Teitelman embodied in the Programmer's Assistant.[12] NLS-SCHOLAR is an artificially intelligent system that can offer computer users stand-alone on-line help ranging from occasional assistance to full tutorial guidance and supervision.

OVERVIEW

NLS-SCHOLAR has the following capabilities:

(a) When used in tutorial mode, it delivers a series of lessons designed for gradual understanding of NLS concepts and commands. Within these lessons, the system pauses to ask the student questions and to propose editing tasks for him to perform using NLS. A student's responses to questions and his performance of tasks are evaluated by the system and if he makes an error, the nature of his mistake is pointed out and appropriate action taken. For example, if a question is answered unsatisfactorily, NLS-SCHOLAR proposes another question of the same kind. If a task is performed incorrectly, depending on the magnitude of the error, NLS-SCHOLAR either resets it for the student to try again, or asks him to proceed and try to fix his mistake, aided by the information NLS-SCHOLAR provides.

(b) The user can formulate requests in relatively unconstrained English. The requests can be questions about NLS concepts or about the state of his work, requests for help in doing a task, or even NLS commands expressed in English. The system is "aware" of what the user is currently doing so that his requests for help can be answered within the context of the problem he is working on. Thus NLS-SCHOLAR not only tells him "The general procedure is . . ." but also "In your case, what you should do is . . .".

(c) NLS-SCHOLAR has the ability to use a person's work space (the NLS file he is currently working on) to show him how to perform editing actions. This gives the system much of the flavor of a human tutor, as if he were taking the student's place at the terminal and saying "Watch me do it for you".

* NLS, the On Line System, is a sophisticated modular system which is being used increasingly as an aid in writing, re-organizing, indexing, publishing, and disseminating information of all kinds.[1] It was developed by Douglas Engelbart and his co-workers at the Augmentation Research Center of the Stanford Research Institute.

** Our system has not yet been interfaced with the real NLS. NLS-SCHOLAR uses LISP-NLS, a partial implementation of NLS's EDIT subsystem written in INTERLISP. The actual interfacing with NLS has been contemplated in LISP-NLS's design, and we hope it will take place in the near future. In the remainder of this paper we shall refer to LISP-NLS as NLS, except where it is important to point out the difference.

*** SCHOLAR, conceived and first developed by the late Jaime R. Carbonell, is an interactive mixed-initiative CAI system dealing with the geography of South America.[7,8] It is capable of answering freely interspersed questions posed by the user in the course of a tutorial session, and it uses teaching strategies similar to those of a good human tutor.[9]

(d) NLS-SCHOLAR is very friendly. Students can ask questions whenever it is their turn to type, make mistakes safely, ask for help doing tasks, and give up and be rescued by the system.

These capabilities allow people to learn from explanation, learn by doing, and learn by asking questions. The tight integration of these capabilities within a working environment makes NLS-SCHOLAR a powerful assistant to its users.

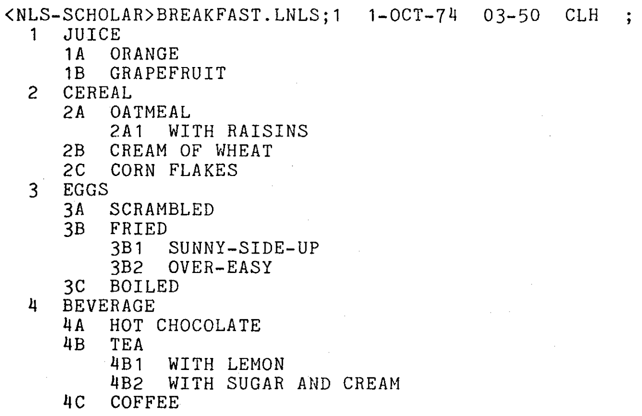
DEMONSTRATING NLS-SCHOLAR'S CAPABILITIES

The flavor of NLS-SCHOLAR is best conveyed by a demonstration protocol which was actually obtained on-line using the latest version of the system. First a few helpful comments:

It is difficult to give a demonstration of a system's capabilities "in vacuo"; questions asked by a student or by the system, as well as tasks proposed and evaluated, arise more naturally and make more sense in the course of a lesson. Since this is a demonstration protocol, our "student" (actually one of the authors) is very obliging and does the appropriate things at the right times to demonstrate specific characteristics of the system.

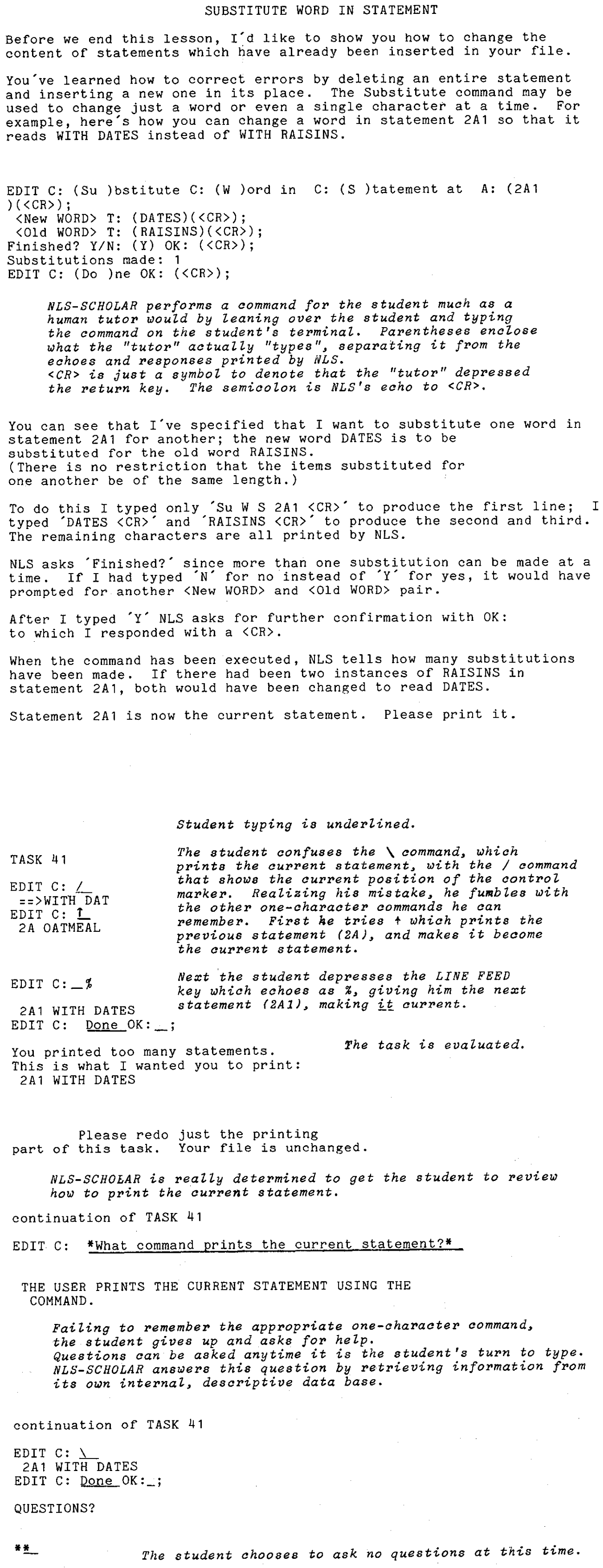
NLS-SCHOLAR uses two bodies of text as its working examples, one a breakfast menu and the other a dinner menu. In the course of a lesson, students learn how to change the contents (and appearance) of these menus by performing editing operations. Menus were chosen as examples because of their direct appeal and general intelligibility, and because the shortness of their entries makes them easy to work with.

In the interest of brevity,* the protocol starts at a point well along in the student's learning of NLS—he has been told about NLS files, how to load them, print them, delete and insert statements, etc. He is about to be taught how to use the Substitute command to effect a change in the breakfast menu.



```
<NLS-SCHOLAR>BREAKFAST.LNLS;1   1-OCT-74   03-50   CLH   ;
  1  JUICE
     1A  ORANGE
     1B  GRAPEFRUIT
  2  CEREAL
     2A  OATMEAL
         2A1  WITH RAISINS
     2B  CREAM OF WHEAT
     2C  CORN FLAKES
  3  EGGS
     3A  SCRAMBLED
     3B  FRIED
         3B1  SUNNY-SIDE-UP
         3B2  OVER-EASY
     3C  BOILED
  4  BEVERAGE
     4A  HOT CHOCOLATE
     4B  TEA
         4B1  WITH LEMON
         4B2  WITH SUGAR AND CREAM
     4C  COFFEE
```

Figure 1: The BREAKFAST File



```
                         SUBSTITUTE WORD IN STATEMENT

Before we end this lesson, I´d like to show you how to change the
content of statements which have already been inserted in your file.

You´ve learned how to correct errors by deleting an entire statement
and inserting a new one in its place.  The Substitute command may be
used to change just a word or even a single character at a time.  For
example, here´s how you can change a word in statement 2A1 so that it
reads WITH DATES instead of WITH RAISINS.


EDIT C: (Su )bstitute C: (W )ord in  C: (S )tatement at  A: (2A1
)(<CR>);
 <New WORD> T: (DATES)(<CR>);
 <Old WORD> T: (RAISINS)(<CR>);
Finished? Y/N: (Y) OK: (<CR>);
Substitutions made: 1
EDIT C: (Do )ne OK: (<CR>);

     NLS-SCHOLAR performs a command for the student much as a
     human tutor would by leaning over the student and typing
     the command on the student's terminal.  Parentheses enclose
     what the "tutor" actually "types", separating it from the
     echoes and responses printed by NLS.
     <CR> is just a symbol to denote that the "tutor" depressed
     the return key.  The semicolon is NLS's echo to <CR>.

You can see that I´ve specified that I want to substitute one word in
statement 2A1 for another; the new word DATES is to be
substituted for the old word RAISINS.
(There is no restriction that the items substituted for
one another be of the same length.)

To do this I typed only ´Su W S 2A1 <CR>´ to produce the first line;  I
typed ´DATES <CR>´ and ´RAISINS <CR>´ to produce the second and third.
The remaining characters are all printed by NLS.

NLS asks ´Finished?´ since more than one substitution can be made at a
time.  If I had typed ´N´ for no instead of ´Y´ for yes, it would have
prompted for another <New WORD> and <Old WORD> pair.

After I typed ´Y´ NLS asks for further confirmation with OK:
to which I responded with a <CR>.

When the command has been executed, NLS tells how many substitutions
have been made.  If there had been two instances of RAISINS in
statement 2A1, both would have been changed to read DATES.

Statement 2A1 is now the current statement.  Please print it.



                  Student typing is underlined.

TASK 41           The student confuses the \ command, which
                  prints the current statement, with the / command
EDIT C: /         that shows the current position of the control
 ==>WITH DAT      marker.  Realizing his mistake, he fumbles with
EDIT C: ↑         the other one-character commands he can
 2A OATMEAL       remember.  First he tries ↑ which prints the
                  previous statement (2A), and makes it become
                  the current statement.

EDIT C: %         Next the student depresses the LINE FEED
                  key which echoes as %, giving him the next
 2A1 WITH DATES   statement (2A1), making it current.
EDIT C:  Done OK: ;

You printed too many statements.          The task is evaluated.
This is what I wanted you to print:
 2A1 WITH DATES


       Please redo just the printing
part of this task.  Your file is unchanged.

     NLS-SCHOLAR is really determined to get the student to review
     how to print the current statement.

continuation of TASK 41

EDIT C:  *What command prints the current statement?*

 THE USER PRINTS THE CURRENT STATEMENT USING THE
  COMMAND.

     Failing to remember the appropriate one-character command,
     the student gives up and asks for help.
     Questions can be asked anytime it is the student's turn to type.
     NLS-SCHOLAR answers this question by retrieving information from
     its own internal, descriptive data base.


continuation of TASK 41

EDIT C: \
 2A1 WITH DATES
EDIT C: Done OK: ;

QUESTIONS?

**              The student chooses to ask no questions at this time.
```

* For a complete demonstration of the present capabilities of NLS-SCHOLAR, see the protocols included in Reference 2.

```
Try the Substitute command yourself by substituting the word
OAT for CORN in the statement containing "CORN FLAKES".


TASK 42

EDIT C: *What statement contains "CORN"?*


THE STATEMENT CONTAINING "CORN" IS:
 2C  CORN FLAKES
```

The student wants to know what the correct statement number is for him to type. The system obtains the answer by actually looking into the student's work space -- the answer is not stored (it cannot be!) in the internal data base.
The system interprets correctly most of the common paraphrases of this type of question, e.g.
"What is the statement containing ..."
"What statement contains ..."
"Where is ...", etc.

```
continuation of TASK 42

EDIT C: Substitute C: Word in  C: Statement at  A: 2C;
 <New WORD> T: CORN;
 <Old WORD> T: OAT;
Finished? Y/N: Y  OK:_;
Substitutions made: 0
EDIT C: Done OK:_;
```

The student switches around the new and the old words, so no change is made.

```
You should have altered the file
so that this part of it
    2B CREAM OF WHEAT
    2C CORN FLAKES
 3 . . .

would look like this:
    2B CREAM OF WHEAT
    2C OAT FLAKES
 3 . . .

But you did not alter the file.

        Please try to repair your
mistake.  The file is as you left it.
```

NLS-SCHOLAR shows the student what the result of this command should have been if he had performed the task correctly as opposed to the result which he obtained. With this information, the student can "debug" his performance of the task.

```
continuation of TASK 42

EDIT C: Substitute C: Word in  C: Statement at  A: 2C;
 <New WORD> T: OAT;
 <Old WORD> T: CORN;
Finished? Y/N: Y  OK:_;
Substitutions made: 1
EDIT C: Done OK:_;

 Now print your result.


TASK 43

EDIT C: Print OK:/C:_;
       2C OAT FLAKES
    3 EGGS
       3A SCRAMBLED
       3B FRIED
          3B1 SUNNY-SIDE-UP
          3B2 OVER-EASY
       3C BOILED
    4 BEVERAGE
       4A HOT CHOCOLATE
       4B TEA
          4B1 WITH LEMON
          4B2 WITH SUGAR AND CREAM
       4C COFFEE
EDIT C: Done OK:_;
```

NLS allows the student to type as little or as much of each command as he likes, providing it is unambiguously denoted. Here he chooses to type all of it since the command is so short.

```
You printed too many statements.
This is what I wanted you to print:
 2C OAT FLAKES


        Please try to print what I
wanted.  The file is as you left it.


continuation of TASK 43

EDIT C: \
 2C OAT FLAKES
EDIT C: Done OK:_;


Good.  The noun 'WORD' refers to the fact that whole words are
involved, not just parts of words; but it does not imply that a single
word only may be provided as either the <New WORD> or the the <Old
WORD>.  Use the Substitute command to change statement 2A1 again, this
time to read WITH BROWN SUGAR AND CREAM instead of WITH DATES and
print your result.
```

```
TASK 44

EDIT C: Substitute C: Word in  C: Statement at  A: 2A1;
 <New WORD> T: WITH BROWN SUGAR AND CREAM;
 <Old WORD> T: DATES;
Finished? Y/N: Y  OK:_;
Substitutions made: 1
EDIT C: Done OK:_;

I wanted you to change
this part of your file:
    2A OATMEAL
       2A1 WITH DATES
    2B . . .

into this:
    2A OATMEAL
       2A1 WITH BROWN SUGAR AND CREAM
    2B . . .

But you changed this part of it:
    2A OATMEAL
       2A1 WITH DATES
    2B . . .

into this:
    2A OATMEAL
       2A1 WITH WITH BROWN SUGAR AND CREAM
    2B . . .


        I have restored the file as it was
at the beginning of this task.  Please start over.



TASK 44

EDIT C: *How does one do this?*
```

The student is lost and asks for help.

```
 THE USER TYPES 'SUBSTITUTE ', FOLLOWED BY 'WORD ',
'STATEMENT ', '2A1', <CR>, 'BROWN SUGAR AND CREAM',
<CR>,
'DATES', <CR>, 'Y', <CR>, '\', A <SPACE>, 'DONE ',
AND <CR>.
```

NLS-SCHOLAR can show him what to do, as well as...

```
continuation of TASK 44

EDIT C: *Please do it for me*
```

...actually do it for him!

```
EDIT C: (Substitute ) C: (Word ) in  C: (Statement ) at  A: (2A1
)(<CR>);
 <New WORD> T: (BROWN SUGAR AND CREAM)(<CR>);
 <Old WORD> T: (DATES)(<CR>);
Finished? Y/N: (Y)  OK: (<CR>);
Substitutions made: 1
EDIT C: (\ )
 2A1 WITH BROWN SUGAR AND CREAM
EDIT C: (Done ) OK: (<CR>);

QUESTIONS?

*NLS*
```

The student can summon NLS in order to practice (or just to play) without fear of messing up his work.

```
...USING A DISPOSABLE COPY OF YOUR CURRENT FILE

EDIT C:  Print  OK:/C:  Statement at A: 2B;
 V: _;
2B  CREAM OF WHEAT
EDIT C: Substitute C: Character in  C: Statement at  A: 2B;

 <New CHARACTER> T: U;
 <Old CHARACTER> T: EA;
Finished? Y/N: Y  OK:_;
Substitutions made: 2
```

The student replaced the characters EA for U.

```
EDIT C: \
2B CRUM OF WHUT
```

```
EDIT C: Print  OK:/C: Branch at  A: "EGGS";
 V:_;
  3  EGGS
     3A  SCRAMBLED
     3B  FRIED
         3B1  SUNNY-SIDE-UP
         3B2  OVER-EASY
     3C  BOILED
EDIT C: Substitute C: Character in  C: Branch at  A: 3;
 <New CHARACTER> T: *;
 <Old CHARACTER> T: -;
Finished? Y/N: Y  OK:_;
Substitutions made: 3
EDIT C: Print OK:/C:  C: Branch at A: 3B;
 V:_;
 3B FRIED
    3B1 SUNNY*SIDE*UP
    3B2 OVER*EASY
EDIT C: Done OK:_;

...THROWING AWAY THIS COPY

                 The student leaves NLS returning to the
                 'QUESTIONS?' level.  The "Transparent overlay" on
                 which he has scribbled disappears without trace.

*Print branch 3B, please*                  Commands can be issued in
                                           natural language.

...USING A DISPOSABLE COPY OF YOUR CURRENT FILE

EDIT C: (Print ) OK:/C: (Branch ) A: (3B) (<CR>);
V: (<CR>);
  3B FRIED
     3B1 SUNNY-SIDE-UP               The "Tutor" demonstrates how to do it.
     3B2 OVER-EASY                   Notice that the file is in its original
EDIT C: (Done ) OK: (<CR>);          state.

...THROWING AWAY THIS COPY
```

Readers familiar with NLS may fail to recognize it as the system depicted in the protocol. This is because NLS-SCHOLAR teaches the use of a newly emerging version of NLS not yet generally available.

REPRESENTATION OF KNOWLEDGE

Much of NLS-SCHOLAR's knowledge is derived from data stored in a semantic network, and from a set of built-in routines that manipulate and retrieve that data in response to queries. The semantic network is a data base of descriptive information represented in attribute-value format. It contains descriptions of actions and their purposes, descriptions of the procedures necessary to accomplish those actions, and descriptions of their effects and consequences. For example, the semantic network contains the description of the purpose of the Delete command as well as the description of the procedure for its use. English renditions of these attribute-value representations are: "The purpose of the Delete command is to delete a structure unit", and "The procedure (for deleting a structure unit) is for the user to type the word DELETE, followed by the name of the structure unit, the address, and two carriage returns".

The semantic network also contains many other kinds of representations, among them the definitions of concepts, the interrelationships between concepts (such as that a statement is an instance of a structure unit), and the sequence of commands necessary to perform each task correctly.

The retrieval routines, taking a user's query as their starting point, look into the semantic network seeking information relevant to that query. For example, if a user wants to know what the line-feed command is used for, his question would translate into a query that would essentially mean: "Find the purpose of the line-feed command". The retrieval routines would attempt several different matching procedures that would eventually yield something like: "The purpose of the line-feed command is to print the next statement".

The retrieval process is assisted by built-in "reasoning" strategies which are called upon when the matching procedures fail. In fact, in many cases the desired information is not directly stored, but can be inferred from available information. For example, if the query were for the procedure for deleting a statement, the matching procedures would fail. However, the retrieval system would still be able to derive the answer via simple deductive inference: it knows that a statement is a kind of structure unit, and it knows how to delete structure units, therefore it can derive the procedure "Type 'DELETE', followed by 'STATEMENT', . . . ".

Mechanisms such as the ones just described are the seat of the abstract "thinking" abilities of NLS-SCHOLAR. As such, they are not yet very powerful, and much can be done to improve them.* However, it is important to stress here that there is more to "intelligence" than powerful manipulation of symbols.

People's intelligent behavior is not based solely on internal representations and conceptualizations and their attendant reasoning procedures. People's data bases are not only in their memories, neither are their retrieval "routines" solely introspective. We use the world as a data base, and our senses to retrieve information from it. I don't need to have in my head a representation of what is behind my chair; if I need to know, I can just turn around, look, and see!

Because NLS-SCHOLAR deals with a "world" (the world of NLS) with which it shares much of its own being (i.e., it is a computer program that deals with another computer program), it was relatively easy to endow it with some of this latter kind of "intelligence". For example, to make NLS-SCHOLAR "aware" of the state of a user's work, all we had to do was design it so that it could couple with NLS and use it as a sort of sensor of the "world" of the user's file.

This coupling of two systems (NLS-SCHOLAR and NLS itself) constitutes an exceedingly powerful tool. First, it makes it possible for the user to ask questions not only about definitions, descriptions of procedures, etc. (such as "What does back statement mean?", "What command prints the back statement?", or "How do I print a file?"), but also about the ever-changing state of his work (such as "What is the content of statement 3A?", or "Where is the CM now?" or "Print just branch 3 for me"). Thus, in addition to searching for answers in a static semantic network we gain the ability to interrogate the dynamic "NLS world" as well.

Second, this coupling provides an easy way of performing a type of "if-then" inference that would be very hard to perform deductively. Suppose a user asked something like

> "If I deleted statement 2B, what would then be the statement number of the statement containing "CORN"?

Finding the answer by deductive reasoning is possible but difficult. Obtaining the answer by using NLS and "sotto

* Much work has been done on this problem in the SCHOLAR system dealing with the geography of South America.[9]

voce" deleting statement 2B and then seeing where the statement containing "CORN" ends up illustrates a powerful use of this coupling.*

Third, it becomes possible to propose problems or tasks to a student and to evaluate his solutions in an interesting way. All the system has to do is access the correct sequence of NLS commands for the task, perform them on a fresh copy of the student's file, and then compare the results.

Lastly, NLS-SCHOLAR can use its semantic network and reasoning routines to infer a procedure (such as how to delete a statement), use this information to construct an NLS command, and then execute that command. Thus it is able not only to describe procedures but also to synthesize NLS commands using this knowledge.

OVERALL ORGANIZATION

The overall organization of NLS-SCHOLAR is represented in Figure 1. There is an EXECUTIVE which controls and supervises the main functions of the system (question answering, question asking, text delivery, and task monitoring), services their requests, and provides communication paths among them. When in tutorial mode, EXECUTIVE is driven by an AGENDA containing general instructions of what to do next (deliver text, perform a task as if a tutor were demonstrating how to do it, answer questions, evaluate a student's answers, etc.).

TASK MONITOR decides how to call NLS. It can simply allow users to type their commands directly into LISP-NLS, it can make use of "tutor typing" of commands either retrieved from the data base or synthesized by Q/ANSWER, or it can have these commands executed invisibly to the user.

Q/ANSWER is a facility for responding to a user's requests. Q/ANSWER responds not only to questions whose answers are static (i.e. retrievable from the semantic network as in "Give me some examples of printing commands"), but also to questions which refer to what a user is doing and which have answers that are dynamic, i.e., that change with time. For example, the question

WHERE IS THE "CORN"?

must be interpreted as a call to NLS to find the address of the word "CORN" as it exists in the current file. To do this, Q/ANSWER has to synthesize the appropriate NLS commands

Jump Statement 0 ⟨CR⟩
Jump "CORN" ⟨CR⟩

have the context manipulation machinery save the user's environment, perform the commands invisibly, restore the user's environment, and hand back the result of executing these commands to Q/ANSWER which then generates a response.

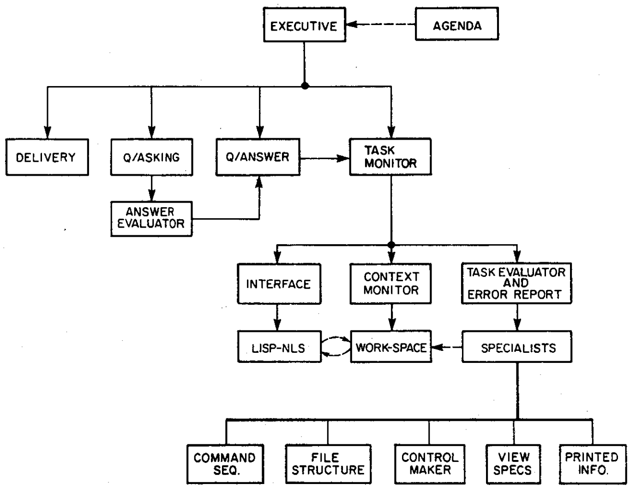


Figure 1

The parser in Q/ANSWER is an adaption of the top-down, semantically directed parser described by Burton,[15] augmented with capabilities that analyze and label the Case relationships[3,16] existing between the main verb and the noun phrases of an input request. In addition it determines the general category that the request falls into (a request for a definition, procedure, address of some word in the current file, etc.). For example, the question

HOW DO I PRINT BRANCH 3A?

parses into the form

```
(QFIND/PROCEDURE ((AGENT USER)
                  (VERB PRINT)
                  (OBJ BRANCH (ADDR 3A))))
```

Thus the interpretation of a request is a LISP function which can then be evaluated (executed) to retrieve an answer. That is, QFIND/PROCEDURE is a LISP function that takes a Case parsed sentence as its input, retrieves the correct response from the data base, and outputs the answer. In our case, the answer is:

THE USER PRINTS BRANCH 3A USING THE PRINT COMMAND. PROCEDURE: HE TYPES 'PRINT', FOLLOWED BY 'BRANCH', '3A', ⟨CR⟩, THE VIEWSPECS, AND ⟨CR⟩.

Only the general procedure for printing a structure unit is in the semantic network. Q/ANSWER instantiates elements specific to the request, in this case the specific structure unit 'BRANCH' and the specific address '3A' (but not the

* Winograd[14] alludes to this inferencing mechanism in one of his SHRDLU protocols ("can a pyramid support a pyramid?" "I DON'T KNOW" "stack up two pyramids" "I CAN'T", and Brown[11] uses it to the fullest in his SOPHIE system.

specific viewspecs because they were not stated in the request).

Q/ASKING is a facility that generates questions for the user to answer. (Although this is not demonstrated in the protocol, the system occasionally asks questions as well as answering them.) If the student's answer is found to be correct, the lesson continues. If it is only partially correct, the part in error is pointed out and another question of the same type is generated. The student does not have to answer the question immediately; he can ask questions and obtain answers from the system, which will then remind him of the question it asked that is still unanswered. The following sample dialogue illustrates this facility. (Parts typed by the user are underlined.)

WHAT STATEMENTS ARE IN BRANCH 3B?
* 3B1 and 3B2*
CORRECT. HOWEVER, YOUR ANSWER IS
INCOMPLETE.
3B1 AND 3B2 ARE O.K.
YOU COULD ALSO INCLUDE 3B.
WHAT STATEMENTS ARE IN BRANCH 1?
Give me a definition of a branch
A BRANCH IS A STATEMENT PLUS ALL ITS SUBST'ATEMENTS.
NOW PLEASE ANSWER MY QUESTION.
* 1, 1A, and 1B*
VERY GOOD

Observe that only the question type is preselected; Q/ASKING knows how to select valid branches right out of the user's file, and to generate questions accordingly.

TASK EVALUATOR and ERROR REPORT base their analysis of a completed task on a state vector of five components:

(1) The commands used by the student.
(2) A representation of the structure and contents of the file.
(3) The position of the control marker.
(4) The state of the "viewspecs" (what kind of a "viewing filter" was used to view the file).
(5) What the student actually printed.

Each of these components defines a "specialist-reporter": a program that knows how to diagnose errors within its specialty and how to indicate to the student what was wrong. For example, the File Structure specialist-reporter compares the result of a student's execution of a task with the correct result and classifies the discrepancies into three cases: something missing, something extra, or something that has a different content. In each case, only enough information to show the discrepant parts is shown to the student ("I wanted you to change xx into yy, but instead you changed zz into tt").

CONCLUSIONS

As computer systems grow in power, sophistication, and complexity, it becomes more and more difficult to become (or even remain) an expert in their usage. Many users prefer sticking to the outdated but familiar facilities offered by a new upward compatible system rather than learning to use the new, more powerful facilities. With the advent of large, geographically dispersed computer facilities, it becomes more and more difficult to get hold of the resident expert and ask him to look over one's problems. There is a real need for something to take these experts' places. We believe that the class of "intelligent" on-line assistants and tutors of which NLS-SCHOLAR is a prototype are a promising solution to this problem.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the work of Alan G. Bell who wrote LISP-NLS, of Gregory Harris who was responsible for the specialists-reporters, and of Joseph Passafiume who wrote the question asking module. This work would not have been possible without the contagious enthusiasm that emanated from countless hours of discussion with John Seely Brown. To him, a very special "thank you."

This work was sponsored by the AFSC-ESD of Hanscom Field AFB under Contract No. F19628-74-C-0088, and by the Personnel and Training Research Programs, Psychological Sciences Division, Office of Naval Research, under Contract No. N00014-71-C-0228, NR No. 154-330.

REFERENCES

1. *TNLS Users' Guide*, November 1973, obtainable from Augmentation Research Institute, Menlo Park, California 94025.
2. Grignetti, M. C., L. Gould, A. G. Bell, C. L. Hausmann, G. Harris and J. J. Passafiume, *Mixed-Initiative Tutorial System to Aid Users of the On-Line System (NLS)*, BBN Report No. 2969, November 1974.
3. Grignetti, M. C. and E. H. Warnock, *Mixed-Initiative Information System for Computer-Aided Training and Decision-Making*, ESD-TR-73-290, September 1973.
4. Goldberg, A., *Computer-Assisted Instruction: The Application of Theorem-Proving to Adaptive Response Analysis*, Technical Report 203, May 1973. Psychology and Education Series, Stanford University.
5. Kimball, R. M., *Self-Optimizing Computer-Assisted Tutoring: Theory and Practice*, Technical Report 206, June 1974. Psychology and Education Series, Stanford University.
6. Barr, A., M. Beard, R. C. Atkinson, *A Rationale and Description of the Basic Instructional Program*, Technical Report 228, April 1974. Psychology and Education Series, Stanford University.
7. Carbonell, J. R., "AI in CAI: An Artificial-Intelligence Approach to Computer Assisted Instruction," *IEEE Transactions on Man-Machine Systems*, MMS-11, New York, December 1970.
8. Carbonell, J. R. and A. M. Collins, *Mixed-Initiative Systems for Training and Decision-Aid Applications*, ESD-TR-70-373, November 1970.
9. Collins, A. M., E. H. Warnock and J. J. Passafiume, "Analysis and Synthesis of Tutorial Dialogues," in *Advances in Learning and Motivation*, Vol. 9, G. H. Bower, Ed., Academic Press, in press.
10. Brown, J. S., R. R. Burton and A. G. Bell, *SOPHIE: A Sophiscated Instructional Environment for Teaching Electronic Troubleshooting (An Example of AI in CAI)*, BBN Report No. 2790, March 1974.
11. Brown, J. S. and R. R. Burton, "SOPHIE: A Pragmatic Use of

Artificial Intelligence in CAI," *Proceedings of the National ACM Conference*, San Diego, California, November, 1974.

12. Teitelman, W., et al, *Interlisp Reference Manual*, Chapter 22, Bolt Beranek and Newman and Xerox Corporation, 1974.
13. Carbonell, J. R. and A. M. Collins, "Natural Semantics in Artificial Intelligence," in *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford University, 1973. Reprinted in the *American Journal of Computational Linguistics*, 1, 1974.
14. Winograd, T., *Understanding Natural Language*, Academic Press, 1972.
15. Burton, R. R., "A Semantically Centered Parsing System for Mixed-Initiative CAI Systems," paper presented at the *Association for Computational Linguistics Conference*, Amherst, Massachusetts, July 1974.
16. Fillmore, C. J., "The Case for Case," in *Universals in Linguistic Theory*, (eds.) Bach and Harms, Holt, Rinehart and Winston, 1968.

Area Director:
Richard G. Mills
First National City Bank
New York, New York

User requirements

The issue

Changed priorities and shifts in the Nation's economic and social environment have uncovered in the private sector a chronic and crippling shortcoming in the linkage between the developers and the consumers of technology. If major dislocations and unaffordable inefficiences in our economic system are to be avoided, this gap must be closed.

The essential lacks are (1) means by which exploiters/consumers of technology and technology-based products and services (who, after all, pay the bills) can specify requirements—the precise character and dimensions of needed technologies, and (2) willingness and ability of technology suppliers to meet those requirements in a timely, cost-effective fashion. Perhaps what is needed is a kind of "Market-research" feedback from, for example, the population of commercial-banking computer users to the computer manufacturers that is effective in influencing the characteristics of the products that are subsequently offered by those vendors to that set of users.

Even more troublesome is the need for a more effective communication path from the employers of the manpower "product" of the college/university system to the decision-makers in that system that allows those bill-paying employers to specify the educational backgrounds and skill sets that they currently require.

In the past, the effectiveness of technology users in "ordering up" needed technologies has been very poor. The suppliers have rushed in to fill the void, and for thirty years our direction and rate of development has been determined almost solely by the "push" of the technology developer, not the "pull" of the exploiter. The consequent waste of money, manpower and time has been high. Today such waste must be judged intolerable, yet it threatens to rise. Technology users must formulate and present requirements, and technology producers must respond to them. The problem is, how shall this change be brought about?

The sessions

This complex and difficult problem is treated in two parts. An initial session will offer two submitted papers, each addressing a particular rather specific technology-transfer issue. The second part will be a panel discussion among four distinguished and authoritative leaders in the management of technology. Because of the breadth of view of the panelists, there will be no explicit attempt to link the papers and the subsequent panel discussion.

The papers will be the following two: "A Functional Approach to Turnkey System Procurement," by Wayne Churchman, University of Dallas, Dallas, Texas; and "Why Things Are So Bad for the Computer-Naive User," by William C. Mann, USC Information Sciences Institute, Marina Del Rey, California.

The second session of the two-session Area will be a panel discussion among four distinguished men from various fields involving management of technology and of the process of its creation. This should be a stimulating and authoritative attack on the broad issue outlined above.

The panelists will be Lewis M. Branscomb, Vice President and Chief Scientist of IBM Corporation, Armonk, New York; Chalmer G. Kirkbride, Science Advisor to the Administrator, ERDA, Washington, D.C.; William F. Miller, Vice President and Provost, Stanford University, Stanford California; George R. White, Vice President Corporate Product Planning, Xerox Corporation, Stamford, Connecticut.

The intended audience

The two sessions in this area address, from both a specific and a profoundly general viewpoint, a problem that has reached the proportions of a National issue. The "paper" session will treat specific topics from the computer technology/computer applications field, while the broad-guage and senior members of the panel have been invited to use the NCC as a platform from which to address the general issues of technology transfer.

Every individual whose activities involve the development or use of new technology will be affected by what may become the first major reshaping of developer/user interplay since the days of World War II. Developers and users of computer-related technology will have an opportunity to anticipate the impact of this reshaping on their specific areas of endeavor.

Why things are so bad for the computer-naive user

by WILLIAM C. MANN
USC Information Sciences Institute
Marina Del Rey, California

This paper is about people's use of computers to get work done. Computers are tools that operate on symbols. Unlike many other tools, such as eggbeaters, telephones and automobiles, most people in our culture seem to regard computers as alien, mysterious and inherently difficult to use. While some of this view is based on mere hearsay, it is nonetheless held by many who have had some experience with computer systems.

Why? Because many computer systems *are in fact* alien, mysterious and inherently difficult to use. This is true even for interactive systems, where the opportunities for accommodation of users are the greatest and the technical history is the richest.

When people first attempt to use computers, they find that they must communicate with them. They are confronted with a variety of "interfaces," each possessing a "language" and some conventions for abbreviation, spelling help, prompting and the like. There are too many such interfaces, and they are all different in arbitrary, unimportant and hard-to-remember ways, but these are superficial problems. A couple of iterations of thoughtful system reimplementation could reconcile them.

Much worse, the interfaces share a common core of methods and demands which is itself alien, unlike any language known to natural man. This problem is not at all superficial—it arises out of fundamental inadequacies in our usual methods for organizing computations and specifying systems. Depending on one's preference, it can be described as a problem in technology or a problem of the subculture that carries the technology. We will describe it here as a technical problem.

COMMAND LANGUAGES

When we look at the facilities which system interfaces typically provide to their users, we find that they are almost exclusively *command languages*. Users can issue commands and supply command parameters. Other facilities of the interface are designed to aid in the selection and use of these commands. There is very little else.

The problem with command language forms is not that they are formally incomplete; they are not. Neither is it that they limit the users' accesss to the full diversity of the machines, although this happens. Neither are the problems in any fundamental way problems of efficiency or cost.

The basic difficulty, the source of this alienness and intractability, is that commands are an extremely narrow, limiting subset of people's familiar range of expression. Radical specialization is required if a person is to express his desire for the accomplishment of some task entirely as a combination of commands.

THERE'S MORE TO COMMUNICATION WITH PEOPLE THAN JUST COMMANDS

If I wanted you to prepare an index of the books and papers in my office, I would first describe to you how I use them. This would give you some clues which would help you to distinguish a useful index from a non-useful one. I would talk about the urgency of the work, which would give you some clues about how to share your attention between this task and others. Then I would show you how to access the collection, and state whatever properties I expected the final results to have—order, format, index terms etc. At every point I would expect you to acknowledge what you satisfactorily understood, and to discuss and obtain clarification on the rest.

The result of our discussion would not be a complete task specification. Many aspects would remain unspecified. I would expect that as you performed the work, some unexpected choices would become necessary, and that you would make some of those choices yourself based on your knowledge of my goals.

Some Representative Interactive Computing Tasks and Processors*

Task	Processors
Information Retrieval	Dialog (R)
System Manipulation	Timesharing Operating Systems (many)
Programming	APL,PL/1
Text Editing	(many)

* Most of these are PDP-10 programs, some of them peculiar to the ARPANET. But the problem is much more widespread, not confined to one class of machines or one subculture of computing.
Dialog is a trademark of the Lockheed Corporation.

Text Formatting	Runoff, Script
Computer Network Manipulation	Telnet, FTP
Message Processing	Sndmsg, Readmail

Sample Statements from the Two-Person Indexing Task

Goals and Purposes
"I want to be able to find things I have already read, and maintain a list of stuff I haven't read yet by topic, so that I can easily pick things to read."

Examples
"So you would index *Speech Acts* under Searle, and Ordinary-language Philosophy and Elocution."

Description
"The index should include several entry regions, by author and title and so forth, and a region of citations."

Clarifications
"I want all the papers indexed."
"Do you mean the papers in the journals too?"
"No."
"OK."

Hypothetical Conditions
"Suppose I find two different papers with the same title and author."

Functional Descriptions
"The location code tells where the item is physically, within a couple of feet of shelf."

Analogies and Comparisons
"The authors section is like the white pages, and the subject section is like the yellow pages."

Similarities and Differences
"It's like a library card catalog, except that we're not using code numbering or cards."

Refused Commands
"I can't index the papers this week."

I estimate that less than 5 percent or our communication would be in commands, requests or other directives. We would exchange descriptions of objects and processes, convey concepts using examples, and negotiate the meanings of terms. We would discuss hypothetical, perhaps even impossible situations. You might not accept all of my "commands," not even all of the sensible ones. We would talk about goals and purposes for objects, actions and steps in the process. Analogies, similarities, differences and comparisons would all be used to express ideas.

The point of this whole elaborate example is to demonstrate how easy it is to specify an information processing task by using language which hardly includes any commands or command parameters at all. It is even reasonable to expect satisfactory results (from a suitable interpreter) based on such a specification.

Consider the hypothetical examples above, from this indexing task. We can observe several things about them:

1. There is no easy translation for any of them into English commands.
2. They represent kinds of communication that are not commonly provided for at all in people's access to computer systems.
3. They are all different in purpose and apparent effect.
4. They are natural to the task, as done by people.
5. Several of them represent initiatives by the receiver of the task.

The corresponding lessons for design of man-machine communication are:

1. Accommodation of command languages (even an English language command subset) is a strong restriction on people's capabilities. It brings difficulties that are not part of the basic task.
2. Computer systems are commonly deficient in meeting the information needs of users.
3. Using commands is a small part of people's communication repertoire. Since people cannot translate these other kinds of communication into commands easily, (1 above), non-command interaction methods need to be designed into systems.
4. There are lots of different opportunities for major improvement.
5. It might be helpful to give the system more initiative in communication than they typically have in current practice.

It is this diverse kind of communication that is familiar and tractable to the computer-naive person. Most people who could express this task in ordinary language would be unable to reexpress it entirely as a sequence of commands.* The difficulty of translating a task into commands is of course not confined to this example or this class of problems.

For many people, being forced to translate their desires into commands is an unpleasant imposition, and is even felt as an attack on their established competence. For some, the difficulty is never overcome, and for others the use of computers becomes a thing to be minimized, whatever the supposed benefits might be.

DESIGNING SYSTEMS FOR ORDINARY PEOPLE

As a long-range goal, we should seek to develop interaction methods that accept the full range of expression suggested above. Below we consider how to expand our stock of tools in these ways.

* In fact, the computer professionals are the only group who can regularly do it on a variety of tasks. They provide, in the systems they build, tools that are congenial to their own conceptual habits, and alien to nearly everybody else's.

A design approach for today

We need to avoid overly optimistic expectations on systems currently in development—those that are based on command-language communication. Developing command languages and command help facilities is important, but mainly for the computer professionals and heavy users who have already adapted to command language demands.

Merely making easier command languages will not help the really computer-naive users very much, since it does not really address their problems. So in the near term we need ways to supplement the command structures of users' languages. Later we should seek ways to include commands as part of much more comprehensive schemes. We can expect a continuing expansion of the facilities in systems for the computer-naive user. How should that expansion be done?

Three properties are especially important for the systems and programs currently being designed:

1. Language structure that admits non-commands from the user.
2. Intentional imitation of human dialogue at the users' interface. In order to end up accepting a broad diversity of the kinds of communication that people use freely, many steps of imitation will be needed. (This is *not* the same as shifting to natural language. Formal languages can be diversified in the right ways.) We can start now by strengthening the declarative and descriptive parts of languages, allowing alternative equivalent forms of expression and imitating the control structures seen in dialogue.
3. Continuity with the currently available best practice. Although we must move beyond commands-alone, commands will always be with us and be important. There are extreme contrasts in the command styles of today's systems, and there are too many new, badly done interfaces.

Design changes in the future

There is a general lack of the kind of detailed knowledge of human communication that we need. Many of the things that people do in communication, including the entire list indicated above, are not understood well enough to support imitation. We lack a good scientifically established model for the simplest case: successful communication between two cooperating people.

Having a good model of people's communication activities would be beneficial far beyond the sphere of computing. It would be a significant advance for psychology, for education, for the medical knowledge of communication disorders, for documentation and information dissemination, for linguistics and other disciplines as well. It would speak to the strong intellectual interest of our century in epistemology. And it might eventually guide people into more effective communication with each other.

But just on the basis of the potential benefits in computing alone, making direct use of computers feasible and comfortable for broad classes of people, *research in modeling human communication processes deserves a far higher national priority than it currently has.*

There are a few active research projects that are building the right kinds of models of human communication capabilities, in a framework relevant to computer system design. For example, in the automated consultant work at Stanford Research Institute, dialogue between a knowledgeable mechanic and an apprentice is being analyzed, with the intention of allowing similar dialogues to take place between an apprentice and an automated knowledgeable mechanic.[1]

In the Sophie instruction system currently being developed at Bolt, Beranek and Newman, Inc., a trainee in electronic fault diagnosis and repair interacts with a computer program.[2] The program represents and manipulates a circuit with hidden faults. The trainee can discuss hypothetical conditions, seek evaluations of guesses (of faults), cause measurements to be made, and ask questions, all in English.

In the Protocol Analysis System II, developed at Carnegie-Mellon University, a program analyzes a transcript of a student's remarks made while solving a problem, and thereby follows his progress.[3]

The Dialogue Process Modeling work at Information Sciences Institute is building computer-program models of specific two-person interactions in English, analyzing the communication effects that people have on each other.

Each of these efforts involves a computer responding to the language forms that people commonly use. None of them are restricted to command language interaction. Several of them have demonstrated capacity for effective response to a significant portion of English expression.

These projects, and others like them, can multiply our understanding of the details of communication as people do it. The hope is that in the future there will be comfortable communication, with the kind of diversity that human dialogue has, commonly available on computer systems.

REFERENCES

1. Deutsch, Barbara G., "The Structure of Task Oriented Dialogs," Technical Note 90, SRI Project 1526, Stanford Research Institute, Menlo Park, Calif. 94025; appeared in *Proceedings IEEE Speech Symposium*, Carnegie-Mellon University, Pittsburgh, Pa., April 15-19, 1974.
2. Brown, John Seely, Richard R. Burton and Alan G. Bell, *SOPHIE: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting (An Example of AI in CAI),* Bolt, Beranek and Newman, Inc., Artificial Intelligence, Final Report (A.I. Report No. 12), 1 March 1974, pp. 24-25, 96.
3. Waterman, D. A. and A. Newell, PAS-II: *An Interactive Task-Free Version of An Automated Protocol Analysis System*, Department of Computer Science, Carnegie-Mellon University, June, 1973.
4. Winograd, Terry, "A Procedural Model of Language Understanding," *Computer Models of Thought and Language*, W. H. Freeman and Company, San Francisco, California, 1973, pp. 152-186.

Functional approach to turnkey system procurement

by WAYNE CHURCHMAN
City of Dallas
Dallas, Texas

The advent of the low cost minicomputer has extended the computer into many new application areas. Minicomputer systems are used in manufacturing control, laboratory automation, and industrial data acquisition and control applications where the high cost of computers have traditionally prohibited their use. In many of these growing application areas, users want to purchase a fully developed hardware and software package which requires no further development and which provides guaranteed performance. This is the concept of the turnkey system.

Turnkey computer systems are being offered today by a tremendous number of companies. Not only are they offered by computer systems houses and by the manufacturers of minicomputer systems, but also by instrumentation and control systems companies which have traditionally served the markets where these computer systems are now being applied. Most companies in the turnkey systems business have approached a particular application area by developing a standard base of computer hardware and application software which can be adapted to the specific requirements of a particular application. For a given application area, various companies may offer turnkey systems which are very similar from a functional and performance standpoint, but which differ markedly in terms of hardware and software approaches.

A successful procurement procedure for a turnkey system must allow and encourage competitive bidding in which the bidders can take legitimate advantage of their applicable technology in both hardware and software. The procedure must include methods for specifying the system requirements, for selecting the system which best meets those requirements, and for contracting for the implementation of the selected system.

The procurement procedure described here was developed by the City of Dallas, Department of Data Services, for use in the procurement of a minicomputer based real-time data acquisition and control system for the City's water distribution system. The technique is not specific to this application, however, and can be used for any turnkey system procurement.

SYSTEM SPECIFICATION

Specification of the system requirements is the key to any procurement procedure. Many users make the mistake of using a physical specification of the type normally used to specify construction work. The physical specification identifies the minimum physical requirements of the hardware and software in detail and is usually used in conjunction with a low bid system selection. Proposed systems which fail to meet a specified minimum requirement are rejected for non-compliance.

This approach, while it works well for buying concrete, has inherent deficiencies when applied to the procurement of a turnkey computer system. The physical specification fails to consider the system as a single integrated entity, but rather considers each element of the system separately. Total system capabilities, as expressed by such measures as throughput or response time, are more significant than the capabilities of individual component specifications, such as memory cycle time or number of registers.

Physical specifications are often used as a matter of convenience. In fact, most physical specifications are written around one company's hardware and software. Such a specification obviously gives that company a tremendous competitive edge. The user may find that he has effectively created a single source situation with a commensurately high bid price. Even worse is the case where the physical specifications represent an amalgamation of the features of several systems. Either the user gets no bids or he pays the price for special development required by everyone to meet the specifications.

The specification problem can be solved by the use of a functional system specification which places the emphasis on the functional and performance requirements of the system. The functional specification does not omit physical requirements. There is an obvious need for definition of the required system components. Physical requirements are stated, however, as nominal specifications and are intended to provide a framework within which each bidder can propose the best integrated system to meet the overall functional requirements.

The functional specification is not without its own set of problems; problems, however, which have solutions and which are worth the effort to solve. The first problem is that system selection becomes more difficult. If physical requirements are interpreted to be nominal specifications, proposals can no longer be rejected for non-compliance. Low bid system selection must be replaced with a cost/performance evaluation of the proposals. A detailed method for performing the cost/performance evaluation will be discussed later. A second problem is that the flexibility which the functional specification provides during the proposal process is not desirable, in fact is intolerable, as part of the contract for system development. The contract requires better definition than provided by a functional specification. The solution to this problem is a two phase contract in which technical specifications are developed prior to the beginning of system development. This form of contract will be discussed in further detail.

The functional system specification can be organized in a number of ways. The following organization includes all of the important elements of a functional specification:

1.00 *System Concept Summary*
An introduction to the required system and the application for which it will be used. In addition, it is helpful to have a brief statement of work which describes the contractor's responsibilities. Special project requirements, such as user participation in the system development, should be described in this section.

2.00 *Physical Requirements*
A description of the required computer and peripheral equipment. Hardware specifications should be stated as nominal values. It is important to state any preferences the user may have and also to state any alternatives which the user sees as being acceptable. For example the specifications for on-line storage might state that either a fixed-head disc or a moving-head disc or a combination of the two types of storage would be acceptable, as long as they provide the required performance and functional capabilities. This section should specify component packaging requirements and describe where the equipment will be located.

3.00 *Performance Requirements*
A statement of the important system performance parameters. These may include system availability, throughput, response time, expandability, failure mode integrity, and any others which are important to the application. In the case of the water distribution system, the City included a subsection on performance requirements for the man-machine interface.

4.00 *Functional Requirements*
A statement of what the system must do. The functional capabilities described should give the bidder a clear idea of what the system must do without being too restrictive in terms of how the functions are accomplished. Again, preferences or examples can be used, but they should be represented as nominal requirements.

5.00 *Information Requirements*
A definition of the type and volume of data which the system must process.

6.00 *Implementation Requirements*
A detailed statement of the contractor's responsibilities for system implementation. This section should detail requirements for project management, system development, installation, test, maintenance, training, and documentation. The schedule requirements should also be stated in this section. The requirements contained in this section may be interpreted literally, if desired, by including a statement in the request for proposals to that effect. As a matter of practicality, this approach is advisable. If all bidders are required to provide the same implementation services and to meet the same schedule, then the difference in bid prices should reflect only the difference in system capabilities.

Writing a good functional system specification is not easy. The functional approach provides the desirable flexibility to the system specification at the expense of adding the problem of interpretation. There is an increased requirement for responsibility on the part of both the user and the bidder. The user must make sure that the specification honestly represents the requirements of the system he is trying to buy. The bidder has a responsibility to propose a system which meets fully the intent of the system specification.

PROPOSAL SOLICITATION AND EVALUATION

Once the system specification has been completed, it is combined with a request for proposals (RFP) and general specifications to form the bid document. The general specifications will not be discussed in detail here. Suffice it to say that this section of the bid document deals with the general requirements and conditions of the contract and is not specific to a particular system or a particular contract.

The RFP is an important section of the bid document. The RFP must give the bidder a clear idea of the functional nature of the specification and how it will be used in the evaluation of the proposals. It must also specify what information is required in the proposal, where and when proposals will be received, and the conditions of submitting a proposal. Let us turn for the moment to the problem of proposal evaluation.

Many schemes have been proposed for performing a cost/performance evaluation of computer systems. The problem is to come up with a way of relating the capabilities of a proposed system to the bid price of the system so that a single measure of cost/performance is available for comparison purposes.

The most sensible approach to a cost/performance evaluation is one in which the capabilities of proposed systems can be compared in terms of dollars, which can then be directly related to the bid prices. Evaluation techniques which develop some figure of merit for each system in terms of a point total have one glaring deficiency: how is the point total to be related to the bid price?

Proposed systems can be evaluated through the application of evaluation criteria which represent, in quantitative terms, the importance which the user places on specified attributes of the computer system. It is also important to evaluate the vendor and the vendor's capabilities. This can be accomplished through the use of the same technique.

The evaluation criteria identify items to be evaluated and an assigned dollar value for each item. The dollar value represents the maximum penalty which may be assigned for each item. Penalty assignment should consider the projected system life. Penalties may be assigned on a one-time or recurring basis, depending on the nature of the evaluation item, up to the maximum penalty specified for each evaluation item.

The assignment of penalties is based on the existence of some deficiency which will require additional development, purchase, or support; or a deficiency which will limit the useful benefits of the system. Penalty assignments are based on a zero value for the proposal which best meets the specified requirements for each evaluation item, as stated in the system specification. A summation of the assigned penalties for each proposal is used in conjunction with the bid prices to determine the successful bidder.

The specific evaluation items chosen will be different for every procurement but should relate closely to the organization of the system specification. Vendor qualifications can be judged on the basis of the following evaluation items:

A. Experience and Customer Ratings
   1. Installed systems of similar size and nature
   2. Standard computer hardware and software base
   3. Length of experience
   4. Specific application experience

B. Company Resources
   1. Personnel
   2. Facilities
   3. Depth of resources
   4. Financial

C. Support Capabilities
   1. Maintenance organization
   2. Location of maintenance support
   3. Software support

In evaluating vendor qualifications, it is mainly a matter of looking at the risk which may be incurred in doing business with each vendor. Even though the user has a great deal of contractual protection when buying a turnkey system, there are certain irrecoverable costs to the user, due to project delays, if the contractor does not perform. In many cases support capabilities may be the overriding consideration. Support of a system can sometimes be crucial to its success. The user has to evaluate the risk associated with the vendor's continued support of the system over its projected life.

Although no maximum penalties were shown for the evaluation items above, they must be determined and specified along with the evaluation criteria in the RFP. The maximum penalty selected should be consistent with the actual worth of the item and the importance of the item to the user.

The assignment of penalties during the proposal evaluation process is based primarily on information provided in the proposal. The RFP must describe what kind of proposal is required. It is a good idea to specify both the content and the format requirements for proposals. This will make the evaluation task much easier. The following outline is suggested for the RFP:

1.00 *Invitation for Proposals*

2.00 *Definition of Terms*

3.00 *Proposal Requirements, Information, and Conditions*
   .01 General Information
   .02 General Form of Contract
   .03 Inquiries
   .04 Pre-bid Conference
   .05 Examination of Site of Work
   .06 Preparation of Proposals
   .07 Receipt and Opening of Proposals
   .08 Proposal Guaranty
   .09 Proposal Modification
   .10 Withdrawing Proposals
   .11 Irregular Proposals
   .12 Rejection of Proposals
   .13 Disqualification of Bidders
   .14 Statement of Qualifications
   .15 Technical Proposal Content and Format

4.00 *Consideration of Proposals*
   .01 Proposal Tabulation and Evaluation
   .02 Proposal Presentations
   .03 Method of Proposal Evaluation
   .04 Evaluation Criteria

5.00 *Documentation Controls*
   .01 Revision of the RFP and Specifications
   .02 Revision Notice

6.00 *Proposal Forms*

7.00 *Contract Award and Execution*
   .01 Award of Contract
   .02 Retention of Proposals
   .03 Return of Proposal Guaranties
   .04 Surety Bonds and Insurance
   .05 Sureties
   .06 Execution of Contract
   .07 Failure to Execute Contract

8.00 *Contract and Bond Forms*

There are many aspects of the RFP which have not been discussed, but hopefully the outline provided above will indicate some of the necessary elements of an RFP which are not detailed here.

FORM OF CONTRACT

The final important element of the functional approach to turnkey system procurement is the two phase contract. As previously discussed, one of the problems presented in the use of a functional specification is that it does not provide sufficient definition for a contract to develop the system. The selected proposal may lend further definition, but may also contain sections which seem to conflict with the system specification or at the very least lead to ambiguities. There is a need for a single document which specifies in detail the system and the work to be performed by the contractor.

The first contractual phase is designed to provide the necessary definition in the form of technical specifications. Phase I of the contract is the final design phase during which technical specifications are developed by the contractor in cooperation with the user. These technical specifications should satisfy the requirements of the system specification and should be specific to the hardware, software, and services offered in the contractor's proposal.

It should be possible to complete the technical specifications in a fairly short period of time. The City allowed three months to develop the technical specifications for the water distribution system project. User approval of the technical specifications should be contractually required before further work commences on the project. The technical specifications developed during the final design phase, upon approval by the user, become the contractual basis for Phase II, the system development phase of the contract.

The technical specifications are intended to be an amplification or extension of the requirements stated in the system specification and the offerings made in the contractor's proposal. Some requirements may be sufficiently detailed as presented in either the system specification or the proposal and may be repeated verbatim in the technical specifications. Other requirements should be developed during Phase I so that the technical specifications completely describe the hardware, software, and services to be provided by the contractor during the second phase of the contract. The following items should be included in the technical specifications:

- Detailed specifications for all major equipment
- Specifications of all necessary minor equipment
- Final design and drawings for any special consoles or other hardware
- Detailed system flowcharts
- Complete functional descriptions of all computer programs including I/O requirements
- Data base design
- Display and report formats
- Installation drawings
- Detailed test procedures
- Detailed documentation standards

A notice to proceed with Phase II, system development, is issued only after approval of the technical specifications by the user. If for some reason agreement cannot be reached between the user and the contractor on the technical specifications, the contract should contain a provision to pay the contractor for the work done with no further obligation to the user. The payment should be specified as a fixed fee in the contract and should not be excessively high. The contractor should be encouraged to make his profit during Phase II, not Phase I.

CONCLUSION

The functional approach to turnkey system procurement is designed to give the user a method to buy the best system for his application at the lowest price. It is certainly not the easy way out and demands a high level of interaction between the user and the vendor. In many ways, the procurement approach described here is simply a formalization of the interaction that must occur in any successful computer system procurement.

The future of CAM systems

by M. EUGENE MERCHANT
Cincinnati Milacron Inc.
Cincinnati, Ohio

INTRODUCTION

A recent Delphi-type forecast[1] of the future of manufacturing carried out by the International Institution for Production Engineering Research (CIRP) resulted in 94 forecast events on which good consensus was obtained. Of these, 24, or over one-fourth, strongly indicated that the computer-integrated automatic factory would be a full-blown reality well before the end of this century. The three key events which summarize this aspect of that forecast are as follows:

1. By 1980 (median), a computer software system for full automation and optimization of all steps in the manufacturing of a part will be developed and in wide use.
2. By 1985 (median), full on-line automation and optimization of complete manufacturing plants, controlled by a central computer, will be a reality.
3. By 1990 (median), more than 50 percent of the machine tools produced will not have a "stand-alone" use, but will be part of a versatile manufacturing system, featuring automatic part handling between stations, and being controlled from a central process computer.

Thus it is evident that the major event expected to occur in the future of computer-aided manufacturing (CAM) systems is the eventual implementation of such factories.

What are the factors and incentives at work today that provide the motive power for such a major change in manufacturing? One is, of course, the fact that the technology needed for that change now seems feasible because of the rapid growth of the capabilities of the digital computer. However, equally important are the economic and social incentives. Both of these latter factors are contributing strongly to the prospects for early realization of the computer-integrated automatic factory.

ECONOMIC INCENTIVES

Manufacturing normally contributes approximately 30 percent of the gross national product of modern industrialized countries. Yet, in spite of that, manufacturing, although normally thought of as a highly productive and efficient activity, is not generally so. For example, this is clearly true of batch-type metalworking manufacturing, which normally accounts for about 40% of total manufacturing employment. The mass production type manufacturing systems (e.g., automotive transfer lines, etc.) account for less than 25 percent of metalworking parts manufacture. In fact, 75 percent of such parts are manufactured in lots consisting of less than 50 pieces. Carter[2] has revealed that, when the life of the average workpiece in batch-type metal cutting production shops is analyzed, only about 5 percent of its time is actually spent on machine tools and, of that 5 percent, only about 30 percent (or 1.5 percent of the overall time) is actually spent as productive time in removing metal. This result is illustrated graphically in Figure 1. This situation can hardly be called economic or productive. Further, it truly pinpoints the two main areas where by far the greatest improvement in the economy and productivity of metalworking manufacturing can be made today. The first of these is reduction of time of parts in process in the shop, and thus of the resulting extremely high inventory of unfinished parts on the shop floor, and of finished parts waiting for others in process so that assembly of the product can proceed. It is evident from Figure 1 that this inventory could potentially be reduced by up to 90 percent. Resulting reduction of indirect capital and labor costs and improvement of productivity could be enormous. Here indeed is a major incentive to implementation of CAM systems and the automatic factory.

The second area of potentially great improvement is that of percent machine utilization. The 30 percent machine utilization indicated by Figure 1 must be combined with the fact that the average machine spends approximately 50 percent of its time waiting for parts to work on (because of the 95 percent time in transit shown in Figure 1). As a result, the average machine tool in a batch-type shop is being utilized productively (i.e., is actually cutting metal) only about 15 percent of the time. Thus it is evident that this utilization could potentially be increased by 600 percent or more. Resulting reduction of direct labor and overhead costs and increase of productivity could be enormous. Obviously this provides another major incentive to implementation of CAM systems and the automatic factory.

Another major economic consideration today is the rapidly rising cost of manufacturing labor relative to manufacturing productivity. This is illustrated by the data

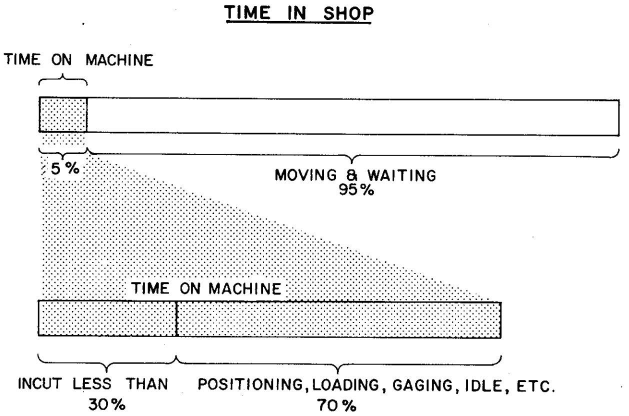


Figure 1—Life of the average workpiece in the average (batch-type production) shop according to Carter[2]

in Table I, for the major industrialized nations of the world, for the period 1965-1970. In the past four years the situation has become even more uneconomic than that shown in the table, due to the even more rapid rise in wages in the current inflationary world economy, and the notable failure of manufacturing productivity to increase at a comparable rate. Quite evidently, this situation can only be reversed by improving the rate of increase of manufacturing productivity, decreasing the degree of labor intensiveness of manufacturing, or both. Both of these can be accomplished by increased implementation of CAM systems and therefore offer an additional major incentive to advancement of such technology.

SOCIAL INCENTIVES

Today, major social factors are also emerging which provide strong incentives for early implementation of CAM systems and the computer-integrated automatic factory. Among these trends, three sets of changing attitudes toward manufacturing are particularly significant, namely those of workers, those of employers, and those of government.

Concerning the first of these trends, there is a steadily increasing reluctance of workers to continue to expose themselves to the manufacturing environment. Thus today, in all the major industrialized countries of the world, there is an increasing shortage of manufacturing workers. This is heightened by the growing opportunities for and rewards in employment in the service industries. This trend is dealt with by Bell[3] in discussing the coming of the post-industrial society. For example, this type of trend occurred first in the field of agriculture. In the United States, as the manufacturing industry developed, the percentage of the work force employed in agriculture declined from 90 percent in 1790 to 4 percent today. Meanwhile, the percentage of the work force employed in manufacturing rose correspondingly during the 19th century. However, in recent years it has begun to decline—from 30 percent of the work force in 1947 to 24.9 percent in 1968. The U.S. Bureau of Labor Statistics projects that by 1980 the percentage will decline further to 22.4, and a Rand Corporation forecast projects that by the year 2000 only 2 percent of the labor force will be employed in manufacturing. Quite evidently, this trend represents a major incentive to implementation of CAM systems for purposes of increasing the automation of manufacturing.

Concerning the second of these trends in attitudes toward manufacturing, namely those of the employers, they are now clearly recognizing the human need for the nature of work to be such as to assure the worker of deep satisfaction from performing it (as well as freedom from unpleasant or harmful conditions). Thus much attention is being directed to methods of accomplishing this. Here the pioneering work of such investigators as Herzburg[4] on job enrichment is proving most useful. Herzburg's significant finding, illustrated in Figure 2, is that, while the so-called hygiene factors of a job (i.e., company policy and administration, supervision, work conditions, salary, etc.) can cause *dis*satisfaction if they are not satisfactory, they can do little to provide on-going job satisfaction. Instead, such satisfaction derives from the adequacy of the so-called motivator factors of a job (i.e., opportunity for achievement, recognition, responsibility, advancement, growth, etc.). The major feature of jobs which provide such opportunities is participation in decision-making. Thus this trend provides a major incentive to implementation of CAM systems for purposes of accomplishing computer-based automation of manufacturing, in view of the endless opportunities it offers for participation in decision-making through interactive type software programs and similar features. In addition, of course, the opportunity it offers for freeing workers from unpleasant, harmful, potentially dangerous or exhausting conditions on the job is tremendous.

The third significant trend in attitudes toward manu-

TABLE I—Rates of Change of Productivity and Labor Costs in Manufacturing

	Average annual percent change, 1965-70			
			Unit labor costs	
Country	Output per man-hour	Compensation per man-hour	National currency	U.S. dollars
Belgium	6.8	8.4	1.4	1.4
Canada	3.5	8.3	4.6	5.1
France	6.6	9.5	2.7	0.6
Germany	5.3	8.7	3.2	4.7
Italy	5.1	9.1	3.8	3.8
Japan	14.2	15.1	0.8	0.8
Netherlands	8.5	11.1	2.5	2.5
Sweden	7.9	10.6	2.5	2.5
Switzerland[1]	6.2	6.2	0.0	0.0
United Kingdom	3.6	7.6	3.8	0.2
United States	2.1	6.0	3.9	3.9

[1] Wage earners only.

Source: Bulletin 1710, U.S. Department of Labor (1971).

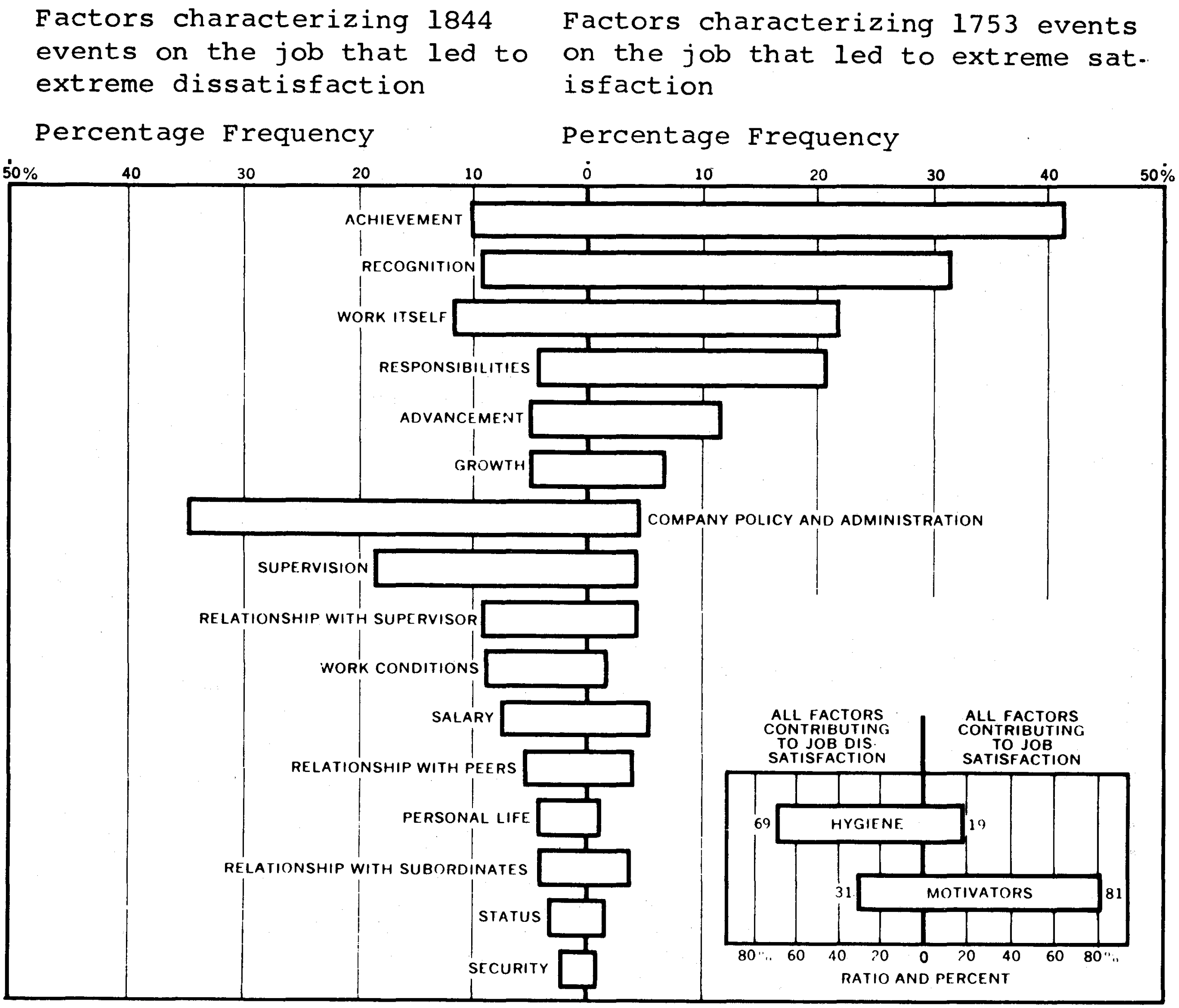


Figure 2—Satisfaction and dissatisfaction factors in jobs according to Herzberg[4]

facturing is the changed attitude which governments throughout the world are taking toward freeing workers from unpleasant, harmful, potentially dangerous or strenuous conditions. In most of the industrialized countries of the world, government is no longer playing the essentially passive role of requiring that, as technology to accomplish improved working conditions is developed, it be put to use. Instead it is now playing the very active role of requiring that such technology be developed. For example, in the United States, the relatively new Occupational Safety and Health Act in effect requires such things as:

1. technology be developed to eliminate the necessity for a worker to ever insert his hand, arm, or any part of his body into a potentially dangerous area of a machine (such as a press)
2. technology be developed to keep the noise level in a factory below 90 dBA (Bollinger[5] has excellently summarized the current status of the international attack on this and other aspects of the problems of noise in manufacturing).

Here again, requirements such as these provide strong incentives to implement CAM systems for automation of manufacturing.

PROGRAMS AND STRATEGY

Most of the industrialized nations of the world today are aware of the foregoing powerful incentives to change the character of manufacturing through implementation of the automatic factory. Likewise, many of them are equally aware of the potential of the digital computer to accom-

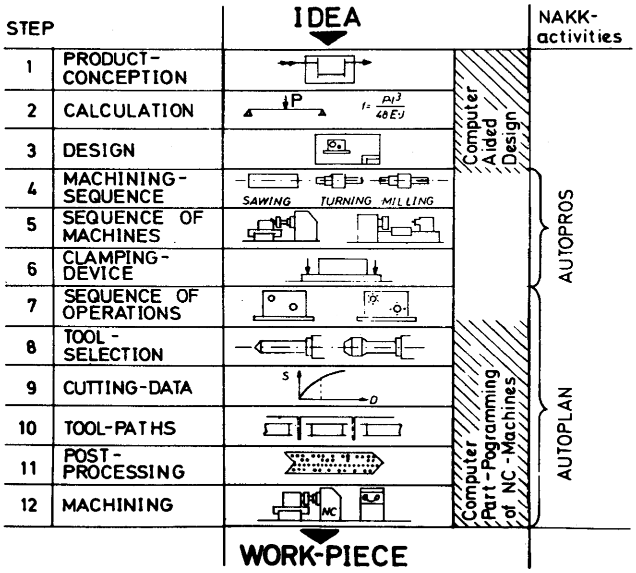


Figure 3—Scheme for development of integrated manufacturing software according to Opitz[6] and Nissen[7]

plish this, and to thereby realize tremendous economic and social national benefits. That potentiality lies in the capability of the digital computer. That capability has powerful potential to provide both the hardware and software components of manufacturing with two essential, powerful faculties, namely:

(1) on-line variable program (versatile) automation
(2) on-line moment-by-moment optimization

As a result, many countries have organized CAM research and development (R&D) programs of considerable scope to try to realize that potential in their own manufacturing industry. The main, long-range objective of these programs is, of course, eventual realization, in that industry, of the computer-integrated automatic factory.

Although realization of the fully computer-integrated automatic factory, through implementation of the concept of the computer-integrated manufacturing system, is the long-range goal of the national programs, it is well realized that to get from today's industrial methods, know-how and installed equipment to that goal requires an evolutionary, rather than a revolutionary, process. The strategy being followed, therefore, is to develop and implement a series of viable, economic steps, in the form of shorter-range programs of R&D on CAM, each having two essential characteristics, namely:

1. Potential for sufficient economic return to justify it by itself and to generate the capital to support development and implementation of the next.
2. Compatibility with eventual attainment of the goal of implementation of the computer-integrated automatic factory.

Out of the variety of such programs being pursued, the following seem to be receiving a major part of the attention and effort, world-wide:

1. Integrated manufacturing software systems
2. Group technology and cellular manufacturing
3. Computer control
4. Multi-station manufacturing systems
5. The computer-integrated automatic factory itself.

Integrated manufacturing software systems

This program of CAM R&D is directed toward development of modular, interfaced, compatible, CAM software systems for on-line computer-based optimization and eventual on-line versatile computer-based automation of complete manufacturing plants. The early economic payoff comes through application of the individual modules, as developed, to increasing on-line optimization of conventional manufacturing plants.

Both Europe and Japan are hard at work developing such software systems for their own use. For example, in Europe, West Germany and Norway are both developing national integrated software systems for metal cutting manufacturing. The "blueprint" for such is the diagram of Figure 3.[6,7] The Japanese have indicated that they are basing their work in this field on pattern processing.[8]

Group technology and cellular manufacturing

This supplement to CAM consists of the layout and organization of a factory and its manpower and equipment on a part-family and product-line basis (rather than a functional basis), using group technology information concerning geometrical and processing similarities between parts to establish the families. This has the eventual CAD/CAM benefit of providing a compatible, economic base for evolution of such factories through increasing use of hierarchical computer control and multi-station manufacturing systems. However, implementation of this technique provides immediate economic benefits from decreased manufacturing lead time, decreased inventory of parts in process, and increased job satisfaction of workers.

This type of program is being actively pursued in both Japan and Europe. Japan is developing its own national part classification system for use in establishing group technology families and is testing it in industry. In the Netherlands, the Metals Institute of TNO[9] is developing, and is testing in industry, a computer-based software system for automatic classification of parts into group-technology families and automatic layout of factories into group-technology cells based on those families. The system will also then carry out automatic production planning for manufacture of those parts and will eventually also provide automatic programming for numerically controlled machining of them. In Britain much R&D and im-

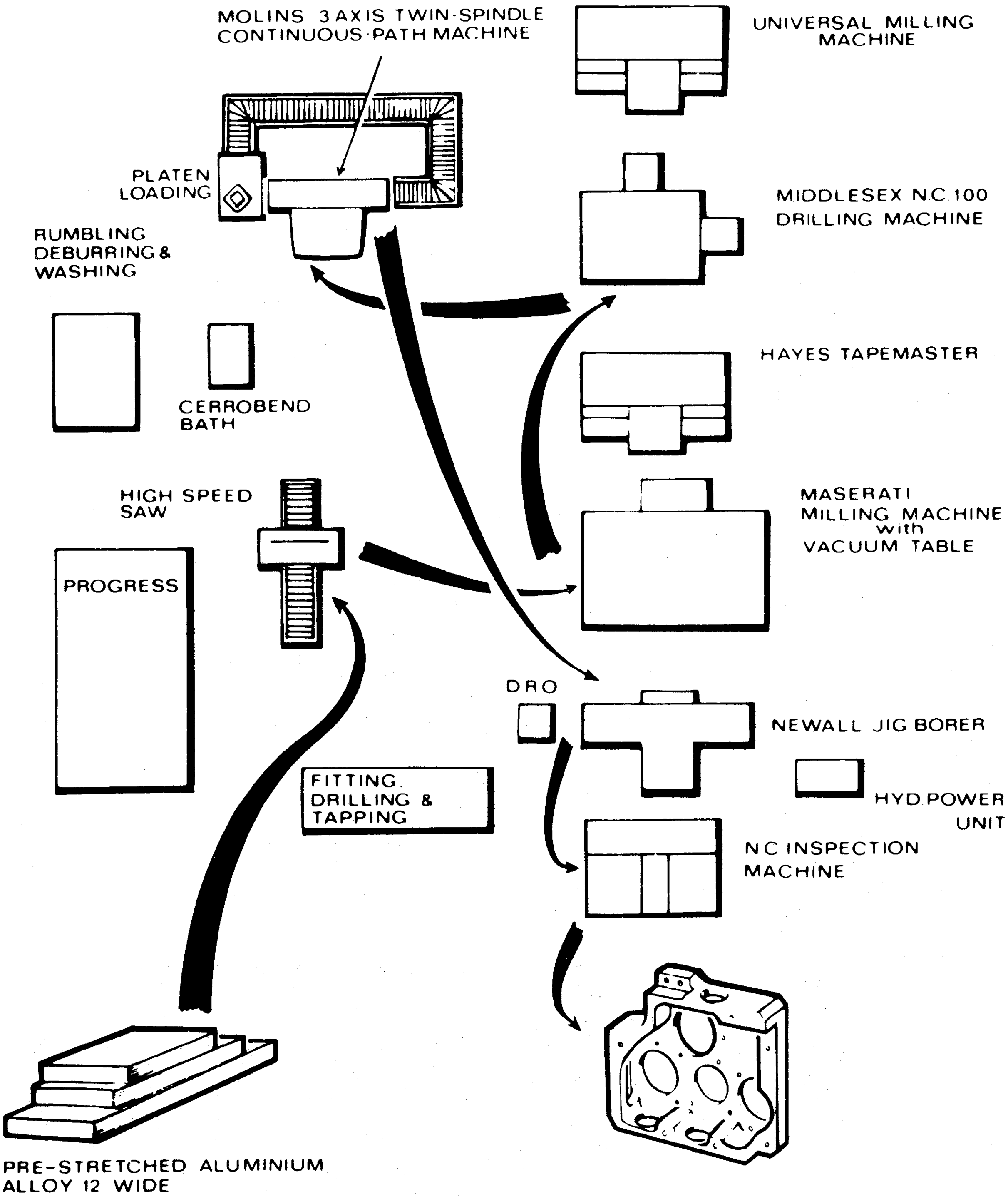

MOLINS 3 AXIS TWIN-SPINDLE CONTINUOUS-PATH MACHINE
UNIVERSAL MILLING MACHINE
PLATEN LOADING
MIDDLESEX N.C.100 DRILLING MACHINE
RUMBLING DEBURRING & WASHING
HAYES TAPEMASTER
CERROBEND BATH
HIGH SPEED SAW
MASERATI MILLING MACHINE with VACUUM TABLE
PROGRESS
DRO
NEWALL JIG BORER
FITTING. DRILLING & TAPPING
HYD.POWER UNIT
N.C INSPECTION MACHINE
PRE-STRETCHED ALUMINIUM ALLOY 12 WIDE

Figure 4—An advanced group technology NC manufacturing cell as implemented by Ferranti[10]

plementation of cellular manufacturing is going on, both in universities and in industry. For example, Ferranti Ltd.[10] in Edinburgh has been busily engaged in implementing such in their manufacture of avionics equipment. Figure 4 illustrates one of their cells utilizing numerical control (NC) for producing a family of box-like parts of the general character of that shown at the bottom right of the figure. By going to cellular manufacturing, Ferranti has reduced the through-put time of parts in process by a factor of approximately 5.

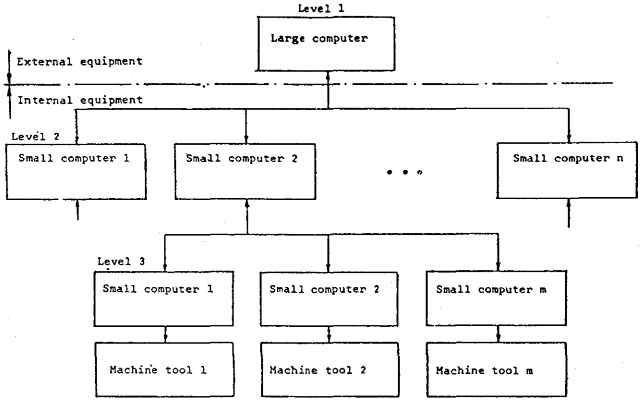


Figure 5—Basic framework of an hierarchical computer control system (i.e., direct numerical control system) for metal cutting manufacturing, according to Bjørke[11]

Computer control

Much of the R&D on this aspect of CAM carried on in Europe and Japan is being given a particular emphasis. This emphasis is directed at evolution of computer numerical control (CNC) and direct numerical control (DNC) in such a manner that, as these bring computer power to the group technology cells on the shop floor (for purposes, initially, of serving only a limited number of NC machines), the computer power is also used to accomplish dynamic scheduling, production control, machine and operator two-way communication with the computer, etc. This encompasses all the machines in the cell and not just the NC machines. Thus, significant economic benefits should become possible very early in the evolution.

This type of program is being carried on very actively in both Europe and Japan. Figure 5 represents the Norwegian[11] concept of the computer hierarchy appropriate to such. As CNC type minicomputer control of the individual machines in a cell is brought to bear at level 3, this in due time provides an economic basis for overall DNC of the cell at level 2 with a larger minicomputer. Eventually it becomes profitable to link all the cells in the factory with a large computer at level 1, providing an initial basis for overall on-line optimization and automation of the factory. Japan is reported to have the largest number of CNC/DNC systems already operating in factories of any country in the world. The number is reported to be on the order of 60.

Multi-station manufacturing systems

This program of CAM R&D involves evolution of the group technology cells in such a manner that, as the percentage of NC workstations operating under CNC/DNC in a cell increases, complete automation and the integration of tool and work handling and transfer within the cell as a whole becomes economically feasible. Such handling and transfer may be done by use of pallets, manipulators, or other means. For example, a metal cutting multi-station manufacturing system or cell, suitable for operation under CNC/DNC, as conceived by Perry,[12] is shown in Figure 6. Thus, in time, each group technology cell is conceived of as evolving into a multi-station manufacturing system operating under CNC/DNC.

Both the European nations and Japan have a number of different types of multi-station manufacturing systems under development. For example, a basic pallet-type system having four different variations for different types of part families has been developed in prototype form in East Germany.[13] However, Japan is reported as having developed the largest number of such systems; the number is said to be approximately 10.

The computer-integrated automatic factory

This program of CAM R&D is of course the final step in the evolution, whereby, through gradual full implementation of a computer hierarchy and an integrated software system, operations in all cells and at other work centers within a plant are dynamically programmed and coordinated for overall on-line optimization and automation of the plant's operations. This includes interfacing the system with computer-aided design in such a manner that initial programming of the automated optimum manufacture of a product is generated in the design stage, with the design optimized for minimum cost of manufacture.

The only nation to date having an announced national plan for accomplishing this by a given date is Japan.[14] Their plan, which goes by the name "Methodology for Unmanned Manufacturing", calls for the development, construction and operation of a prototype "unmanned" machine-building plant by about 1980. The plant would be a 200,000 to 300,000 square foot factory staffed by a control crew of only about 10 persons, as compared to the

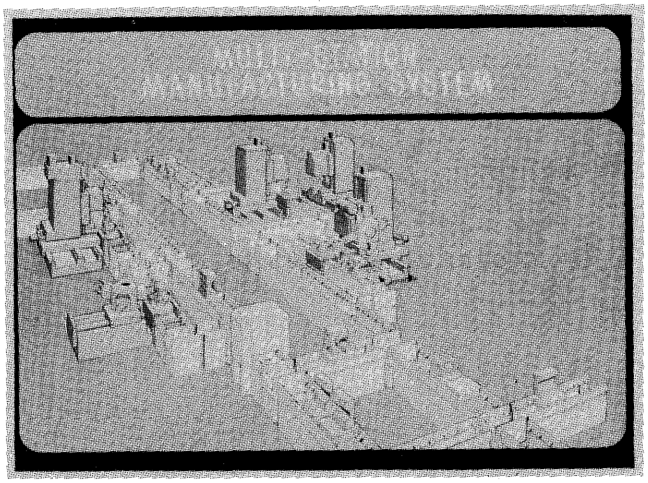


Figure 6—A metal cutting multi-station manufacturing system or "cell" suitable for computer control, according to Perry[12]

normal complement of 700 to 800 workers. The cost of the project will be approximately $100 million.

CONCLUSION

It seems evident that the combination of the powerful technological, economic and social factors and incentives and the concerted national R&D programs active today will indeed make the computer-integrated automatic factory a reality well before the end of this century. Thus all of us who are engaged in advancing the state of the art of CAM systems have exciting and challenging opportunities ahead. Further, in view of the very substantial economic and social benefits which can come from implementation of CAM systems and the resulting eventual realization of the computer-integrated automatic factory, as described above, we have a major responsibility to direct our efforts toward cooperation on a national scale to attain that goal as rapidly as possible.

REFERENCES

1. Merchant, M. E., "Delphi-Type Forecast of the Future of Production Engineering," *CIRP Annals,* Vol. 20, No. 3, pp. 213-225, 1971.
2. Carter, C. F., "Trends in Machine Tool Development and Application," *Proceedings of the Second International Conference on Product Development and Manufacturing Technology,* Macdonald, London, pp. 125-141, 1972.
3. Bell, D., *The Coming of the Post-Industrial Society: A Venture in Social Forecasting,* Basic Book, New York, 1973.
4. Herzburg, F., "One More Time: How Do You Motivate Employees?" *Harvard Business Review,* Vol. 46, No. 1, pp. 53-62, 1968.
5. Bollinger, J. G., "Noise—An Industrial Pollutant of International Concern", *CIRP Annals,* Vol. 22, pp. 197-202, 1973.
6. Opitz, H., "Integrated Information Processing in Industrial Production," *Proceedings, CIRP International Conference on Application of Computers to Manufacturing,* Halwag, Berne, pp. 69-79, September, 1969.
7. Nissen, K. G., "AUTOPROS-Automated Process Planning System," ibid, pp. 47-53.
8. Sata, T. and H. Yoshikawa, *Some Aspects of Pattern Processing for Manufacturing Systems,* unpublished CIRP report, August 28, 1970.
9. Keus, J. A. and B. A. Schilperoort, "Group Technology and Automated Work Preparation for Conventional and Numerically Controlled Machines," *Metaalinstituut TNO Communications,* No. 7, pp. 3-11, February, 1973.
10. Allen, C., *The Formation and Operation of a Cell System,* Paper No. 1001, Fourteenth International Machine Tool Design and Research Conference, University of Manchester Institute of Science and Technology, September, 1973.
11. Bjørke, Ø., *On-line Numerical Control Systems,* Paper No. CPA-05, CIRP Third International Seminar on Optimization of Manufacturing Systems, Pisa, Italy, June, 1971.
12. Perry, C. B., "Variable-Mission Manufacturing System," *Proceedings of the First International Conference on Product Development and Manufacturing Technology,* Macdonald, London, pp. 314-333, 1970.
13. Tschink, K., *New Manufacturing Systems for Small and Medium Batch Production,* paper presented at Fourteenth International Machine Tool Design and Research Conference, University of Manchester Institute of Science and Technology, September, 1973.
14. "Makers Plan Developing Unmanned Machine-Building Plant in Seven Years," *Japan Economic Journal,* June 19, 1973.

Parts representation in CAD/CAM

by IKUO OYAKE
Oki Electric Industry Company Limited
Tokyo, Japan

INTRODUCTION AND BACKGROUND

During this decade, numerical models of mechanical parts have been exploited along with the activities related to computer aided design and manufacturing. Indeed, various types of engineering analysis in design have involved the representation problem of the object. N/C machining has required mathematical expressions of the surfaces to be machined for preparing the control information. Also systematical approaches to production require complete numerical representation of parts to provide efficient and smooth interface between design and manufacturing. This paper will describe one approach to parts representation and its applications.

So far, the term computational geometry has been advocated by A. R. Forrest[1] to denote the representation, synthesis and analysis of shape information. Concerning surface delineation, S. A. Coons[2] pointed out the problem as the one of 'minimum specification' in his theory, P. Bezier[3] has provided an intuitive way of geometric construction, the so-called Bezier polygon, and successfully implemented it in UNISURF.[4] This procedure has been extended by W. J. Gordon and R. F. Riesenfeld.[5] They showed that Bezier's method utilized Bernstein polynomials and, in fact, that the success of his theory is attributable to the use of these polynomials. Moreover, they have developed B-splines which possess the local modification property. Also, the advantage of parametric representation has been emphasized by A. R. Forrest; D. V. Ahuja and S. A. Coon[6] have introduced rational polynomials in expressing conventional curves and surfaces such as a circle, a parabola, a sphere and so on.

On the other hand, complete mathematical models of mechanical parts have been studied in parallel. The possible approaches in the APT environment have been summarized by M. M. Cutterman.[7] N. Okino[8] has exploited the half-space approach and some set operations for parts construction and successfully implemented them in the TIPS-1 system. I. C. Braid and C. A. Lang[9] have developed volume concepts in parts representation and established basic primitives for the procedure of building bricks. The GIL languages, which are extensions of APL, have been developed by M. Hosaka[10] for the design and drawing of 3-D objects. Also, the PADL languages for part and assembly description, including tolerance facilities, have been investigated by A. A. G. Requicha, et al.[11]

The approach which will be explained in this paper aims at the synthesis of various component surfaces and the construction of complete parts representation. The category of component surfaces includes any free-formed surface in parametric form as well as conventional surfaces such as planes and cylinders.

THE APPROACH

Figure 1 depicts the functional aspects of the approach. Generally for the synthesis process, topological data has to be specified to show how each surface adjoins the others. This specification usually requires a great deal of ability in 3-D space recognition. The approach has been designed to minimize this task and replace the topological data by local values associated with each surface. Therefore, the designer need not know the surface boundaries of the final shape. The only data required is an interior point and outward direction for each surface (Figure 2.)

Formulation of 3-D objects

In this paper a 3-D object is a point set in 3-dimensional space which is bounded by a finite number of surfaces. This 3-D object may be characterized by its vertices, edges and n surfaces. Consider Figure 3. The surface SRi is a closed region bounded by some of the other surfaces. Express the region of SRi by the sequence of the edges, e_{i1}, e_{i2}, . . . , e_{ip}, . . . , e_{ik}. Then define as follows:

MAT$\equiv$(Eij), where Eij$=e_{ip}$ if e_{ip} is a common boundary of SRi and SRj; Eij$=\emptyset$ otherwise.

By the definition Eij, the i-th row vector forms the region of SRi. On the other hand, the j-th column vector reveals which other regions adjoin SRj. Of course, in the case of an object, each column vector has the same edge elements as the corresponding row vector. Further, the orientation determined by the outward vector on each surface gives a direction to each edge of the region (Figure 3). Then, the connectedness of an object is equivalent to the fact that the i-th column vectors have the same edge elements but the opposite direction as the i-th row vectors. This property of the incident matrix MAT will identify the topological relationships among the surfaces of the 3-D object.

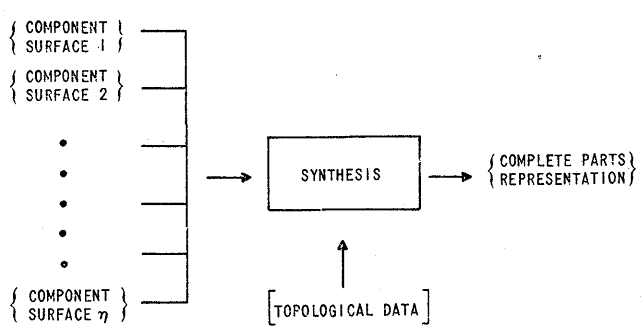


Figure 1—Functional aspects of the approach

Construction of 3-D objects

An n-sided object can be expressed by its set of n faces. The basic idea of the construction is to define the general space of possible solutions and then use an iterative method to find the object.

Step 1. Each component surface has the following parametric equation,

$$\text{SCi} = \text{Fi}(u_i, v_i),\ i \leq n.$$

Let Ni be a point on SCi interior to the final object. Then there exists α_i and β_i such that Ni = Fi(α_i, β_i). Also let Vi be a vector directed outward from SCi, then the integer IP can be determined for later use as follows:

$$\text{IP} = \text{sign}((\partial\text{Fi}/\partial u_i * \partial\text{Fi}/\partial v_i) \Big|_{\substack{u_i=\alpha_i\\ v_i=\beta_i}} *\text{Vi})$$

Step 2. Fix i. Then the set GEOMi = {SCi∩SCp∩SCq | p≠q} consists of a collection of curves (p=i or q=i) and points (p≠i and q≠i). Let CUVi be the set of all curves C in GEOMi such that at least two points in GEOMi lie on C. If C∈CUVi, then C is divided into subarcs by the points of GEOMi which lie on C. Let Ic denote the set of these subarcs. Finally, let ARCi = ∪Ic, and define a graph:

c∈GEOMi

GRAPHi = {I∈ARCi | I adjoins some other

J∈ARCi at each vertex of I}.

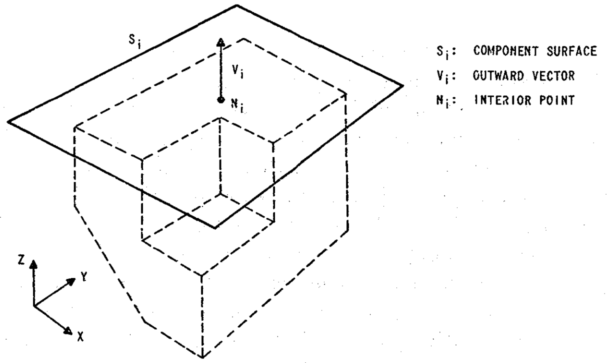


Figure 2—Specifications

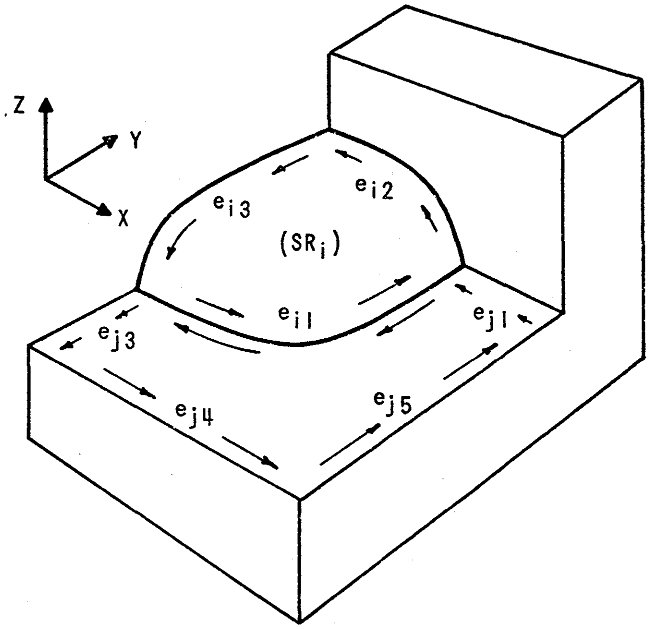


Figure 3—3-D object

Step 3. The topological aspects of GRAPHi are invariant under the mapping Fi^{-1}. Define the graph GPHi in the (u_i, v_i) space as follows:

GPHi≡Fi^{-1} (GRAPHi) = {D | D = Fi^{-1} (I),

I∈GRAPHi} (Figure 4.)

A domain is a closed connected point set in the (u_i, v_i) space bounded by some of the edges in GPHi. If a domain contains no edge of GPHi in its interior, it is called a fundamental domain. BASEi is defined as follows:

BASEi = {M | M is a fundamental domain in GPHi}

Step 4. The boundary curve of M in BASEi is denoted by ∂M, which is a sequence of edges in GPHi. There exists one domain, say Mo, in BASEi which contains (α_i, β_i) in its interior. Then the orientation of Mo can be determined so that the following expression (using the winding number in a

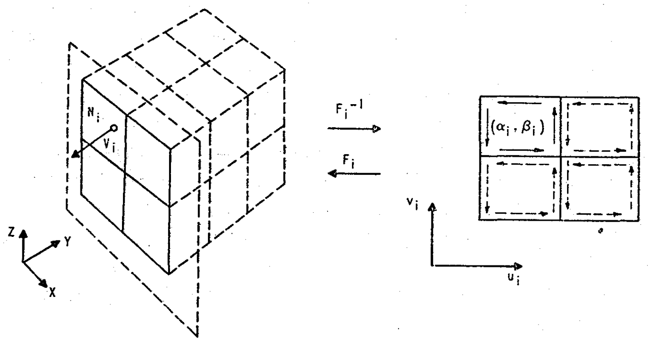


Figure 4—Construction of graphs

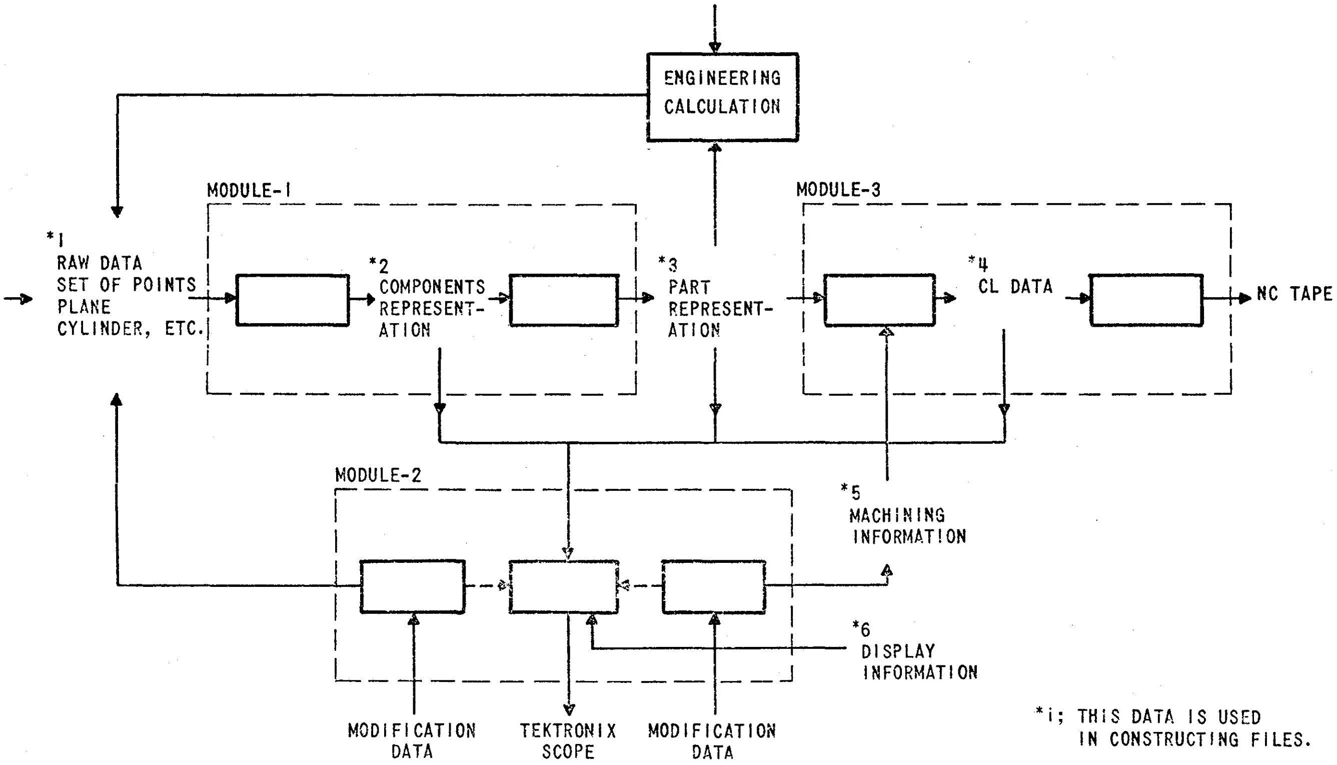


Figure 5—System flow

complex plane) is equal to 1,

$$\text{Real}\ ((2\pi\rho)^{-1}\int_{\partial \text{Mo}}(z-zo)^{-1}\,dz)\ {*}\text{IP},$$

where $z=u_i+\rho v_i$ (ρ is an imaginary unit).

Mo will induce a unique orientation over the remaining fundamental domains (Figure 4).

Step 5. The sum of two fundamental domains can be defined if and only if they have a common edge. The boundary of the sum of Mj and Mk is expressed by:

$$\partial(\text{Mj}+\text{Mk})=\partial\text{Mj}+\partial\text{Mk}-\partial\text{Mj}\cap\partial\text{Mk}.$$

Thus, the orientation of (Mj+Mk) is determined by that of Mj and Mk.

The above discussion may be extended to more than two summations. Any domain, including Mo, can be described by the following summation:

$$\text{DOMj}=\text{Mo}+(\text{M}_{j1}+\text{M}_{j2}+\cdots+\text{M}_{jp}).$$

The set of all the above domains is denoted by DOMAINi.

Step 6. Let Ωi be the following set of regions on SCi,

$$\Omega\text{i}\equiv\text{Fi (DOMAINi)}=\{\text{RAG}\mid\text{RAG}=\text{Fi(DOM)},\ \text{DOM}\in\text{DOMAINi}\}$$

Then the set of general possible solutions will be the product set:

$$\text{SPACE}=\Omega_1\times\Omega_2\times\cdots\times\Omega\text{i}\times\cdots\times\Omega\text{n}.$$

An iterative process is employed to search for a combination ($\text{RAG}_{k1}{}^1$, $\text{RAG}_{k2}{}^2$, . . . , $\text{RAG}_{kn}{}^n$) which satisfies the property of the incidence matrix.

APPLICATIONS

A die making system is one of the applications of parts representation. The overall view of a system in design and manufacturing of die cavities is shown in Figure 5. The file of parts representation is an input to the manufacturing module. In the case of N/C language systems where regional milling capability is available, the additional input information will be a machining area, guiding lines, pickfeed values, clearance planes, a cutter configuration, tool axis and postprocessor related words. A multiclient program, CADDEF, which aims at design and fabrication for sheet metal stampings has been proposed by IIT Research Institute. In this program, finite element analysis on parts geometry is employed for mathematical simulation of die stamping processes.

ACKNOWLEDGMENT

This work has been supported by the management and computer science division of IIT Research Institute. The author is greatly indebted to Dr. Hunter Shu, who has provided constructive suggestions. Also, the author wishes to thank Dr. Kenneth Fox and Dr. Rodney Kosloski for their kind reviews of this report.

REFERENCES

1. Forrest, A. R., "Computational Geometry," *Proceedings of the Royal Society*, London, A321.
2. Coons, S. A., *Surface for Computer-Aided Design of Space Forms*, Project MAC, MIT, 1964. Revised to MAC-TR-41, 1967.
3. Bézier, P., *Emploi des Machines a Commande Numérique*, Masson et Cie, Paris, 1970. Second revised edition translated to English by A. R. Forrest, to be published by Wiley and Sons, 1972.
4. Bézier, P., *Procédé de definition numérique des courbes et surfaces non mathématique; Système UNISURF*, Automatisme 13, May 1968.
5. Gordon, W. J. and R. F. Riesenfeld, "Bernstein-Bézier Methods for the Computer-Aided Design of Free-Form Curves and Surfaces," *Journal of ACM*, Vol. 21, No. 2, 1974.
6. Ahuja, D. V. and S. A. Coons, "Geometry for Construction and Display," *IBM System Journal*, Vol. 7, No. 3 & 4, 1968.
7. Cutterman, M. M., Final Report, *APT Numerical Description Study*, Prepared for Sandia Corporation, Livermore Lab., Contract No. AT-(29-1)-789, IIT Research Institute, 1964.
8. Okino, N., Y. Kakazu, and H. Kubo, "TIPS-1: Technical Information Processing System for computer-aided design, drawing and manufacturing," *Preprint of PROLAMAT' 73*, Budapest, April 1973.
9. Braid, I. C. and C. A. Lang, *Computer-Aided Design of Mechanical Components with Volume Building Bricks*, Computer Lab., University of Cambridge, England, 1972.
10. Hosaka, M. and F. Kimura, *Computer Processing of Solid Objects*, University of Tokyo, Japan.
11. Requicha, A. A. G., N. M. Samuel, and H. B. Voelcker, *PADL-1.0: Part and Assembly Description Languages*, TM-20, Production Automation Project, University of Rochester, 1974.
12. *Documentation for the CAM-I fourth Sculptured Surface Experimental Release System*, SSX4, IIT Research Institute, 1974.
13. Kishi, H., I. Oyake, Y. Shimura, K. Nagai, and T. Hatta, "OKISURF System," *Proceedings of the 11th Conference*, Numerical Control Society, 1974.
14. *The CADDEF Program—Computer Aided Die Design and Fabrication for Sheet Metal Drawing*, Project Proposal, IIT Research Institute, 1974.

Two application programs which link design and manufacture

by HENRY MERRYWEATHER
Computer Aided Design Centre
Cambridge, England

INTRODUCTION

The Computer-Aided Design (CAD) Centre at Cambridge, England has pursued a policy of cooperation with various industrial organizations to specify and develop computer programs which fulfil a variety of needs. These needs are dependent on the nature of the industry involved and have resulted in diverse application programs which include: visualization packages to assist architects to "see" the building they are designing before it is built; systems to assist the design and costing of chemical plant, and programs to generate the precision artwork for printed circuit boards.

The application programs are developed by groups at the CAD Centre who have a specific responsibility for one sector of industry or one class of industrial problems. One group, the Industrial Engineering Group, is concerned with producing economically and technically sound links between the design and manufacture of components. These links are contained in two packages and the purpose of this paper is to show: the programs overall characteristics and the computer systems on which they are used; the way the packages have been developed and an outline of their facilities, and examples of the way the systems have been used.

INDUSTRIAL ENGINEERING GROUP APPLICATION PROGRAMS

The Industrial Engineering Group application packages are directed towards providing viable solutions to the problem of linking design and manufacture of a wide range of components. However, the provision of links between design and manufacture often encourage different methods of working within an organization and influence other parts of a manufacturing business such as marketing as will be seen when the application programs are discussed in greater detail below. Throughout the development of the packages, industrial participation has been an essential ingredient and it is considered that this participation has been crucial because it ensured that the resulting computer programs do provide the facilities and capabilities required by genuine industrial needs. Further assistance towards the development of viable application programs has been the adoption of at least the philosophy, if not the actual routines, of general purpose program aids, such as those associated with the production of drawings on the various graphical output devices (e.g., plotters and storage tubes) that are used. The assistance of general purpose program aids is also seen as a substantial contributory factor towards successful application programs because it reduces both the cost and time scale involved during the development time of the computer programs. The ability to move the group's programs readily from computer to computer is a characteristic which is adhered to as much as is practicable. The portability of programs is necessary because it is intended that the programs can be supplied to those who require them and there are many different computers which have the necessary languages and operating systems. The use of Fortran IV and adoption of general purpose program aids are seen as two essential ways of assisting program portability because Fortran IV is available on most of the computers which are in all other respects suitable for such programs and as general purpose software often deals with the computer dependent problems such as are found with input and output, these difficulties only have to be overcome once.

Apart from portability, another important criterion in the specification of the Industrial Engineering Group's application packages is that they can be easily tailored for specific problems. This is considered important because, although generalized computer solutions can be written to solve a class of problems, the resulting program is often large and difficult to use.

During the early stages of development, multi-access computers which consisted of a main or host processor and satellite computers were used to provide interactive facilities to pave the way for the more flexible forms of computer power that have come with advances in computer technology. Now, as computer hardware costs have been reduced, relatively cheap but powerful mini computers provide good interactive and economic running of these application programs while still permitting access to main frame computers when necessary. The response from the mini computer is ensured because it will be used in a dedicated manner with often only one user on the machine at any particular time. Thus, the mini computer is now being used for certain Design and Manufacturing problems in the same way that it has been used for some years to control one item or section of process plant.

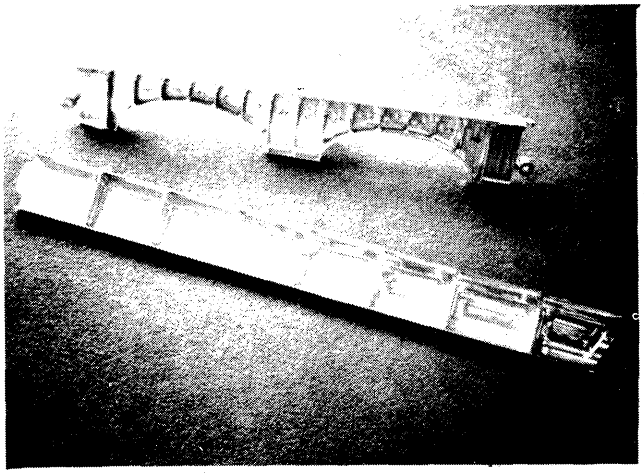


Figure 1—Examples of components suitable for machining by the GNC system

Two main application packages have been developed by the group, namely:

GNC a method of verifying the design, producing n/c tapes and checking manufactured components which are two and two and a half dimensional and

POLYSURF a method of designing, drawing and, where appropriate, producing n/c tapes for three dimensional components particularly those which have doubly curved surfaces.

The remaining parts of this paper explain, in more detail, the characteristics of these application programs.

GNC

Development of GNC

The CAD Centre has always had the ability to produce drawings and therefore was often requested to plot the tool paths of tapes for n/c machine tools. This way of tape checking was found to be of only limited assistance because not all errors could be detected and even when errors were found there was no ready means of making the necessary corrections.

Thus, the concept of the GNC[1,2] (Graphical Numerical Control) package was stimulated by the need to overcome the deficiencies of checking tool paths by plots from tapes for n/c machines. GNC was specified by active participation of three British companies (British Aircraft Corporation (BAC) of Weybridge, Plessey Co., and Midcast Numerical Controls) who not only regularly met with CAD Centre staff but provided assistance in the form of computer programs and n/c machine time. The overall specification of the system was that it should: be suitable for two and two and a half dimensional components (Figure 1); simplify part programming; and reduce tape preparation lead time without incurring extra costs.

It was decided to separate the production of n/c tapes into the three distinct stages of: definition of component geometry; description of machining pattern; post processing and one stage would be checked before the next stage was started. The separation into three stages also meant that the component geometry could be specified by staff other than those usually involved with part programming. Such staff could well be part of the design team of the component who, because they designed the component, could more economically carry out the geometry definition in such a manner that it could immediately be used to describe a suitable machining pattern.

It was decided to use an existing program called KCURVES[3] for the component geometry definition because this was immediately available and portable.

The program to describe the machining pattern was developed at the CAD Centre to be interactive, using the graphic capability of the storage tube terminal (Figure 2) to show the component and cutter path as it was generated. The output from the machining pattern program was arranged to be in a standard form so that whenever possible existing post processors could be used.

As has been shown, the original brief of GNC was to provide a more cost effective method of producing n/c tapes. Successful use of GNC by the participating and other firms has shown that this brief has been achieved. Additionally it has been found that relatively small enhancements to the machining sequence program, enable accurate drawings to be produced using a flatbed plotter which provide both: an additional way of controlling machining because the precision plots can be used for machine tools with optical followers, and a method of checking profiles because the profile of the machined component, e.g., a hob to cut gear wheels can be compared with the accurate drawing.

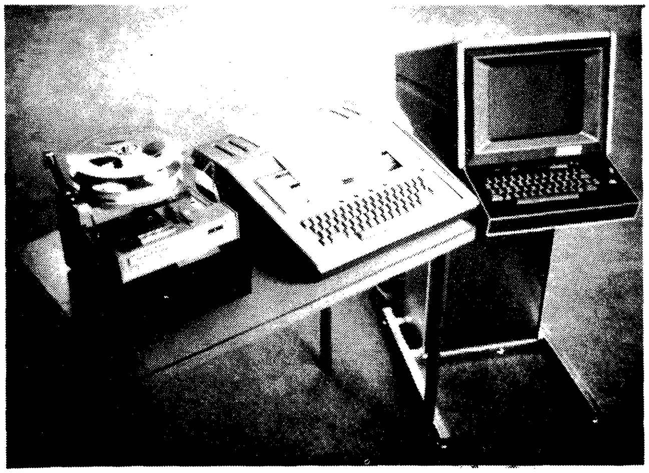


Figure 2—Storage tube terminal with tape reader and punch

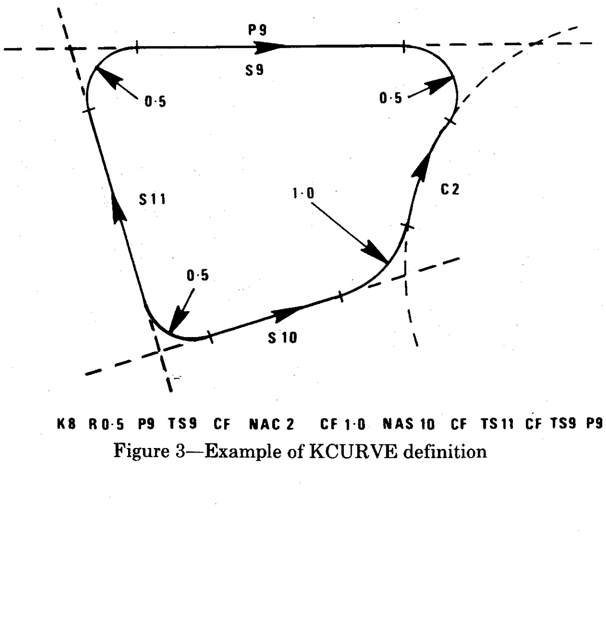
Facilities within GNC

The geometry definition program (KCURVES) allows components to be described as arbitrary combinations of parts of straight lines and segments of circles. The definition is carried out in the two stages of: definition of all the straight lines and circles required and description of how the parts of straight lines and circles are to be used.

Most normal engineering components can be described because at least forty different ways of describing straight lines and circles are available. An example of the way the straight lines and circles are joined together is shown in Figure 3 which also demonstrates the particularly useful feature of being able to insert fillet radii between any two adjacent parts of the complete curve without defining these radii at an earlier time. A further option is the ability to fit a smooth curve through a series of points with a number of circles; this is particularly useful when profiles, such as aerofoil and cam shapes, are defined as a table of points.

The machining pattern program is interactive and the way this characteristic is used is demonstrated more clearly in the next section which describes GNC in use. However, the main features of the program are listed below:

- automatic scaling and drawing of component curves for checking and reference
- translation, rotation, reflection and repetition of geometry
- calculation and display of offset profiles for use in cutter radius compensation
- the ability to define part programming statements to describe cutter movements and associated data
- animation of the tool movements in three orthogonal and isometric views
- editing the part program by inserting new statements and deleting those no longer needed
- reprocessing the part program to produce APT-compatible cutter location data.



K8 RO·5 P9 TS9 CF NAC 2 CF 1·0 NAS 10 CF TS11 CF TS9 P9 EK

Figure 3—Example of KCURVE definition

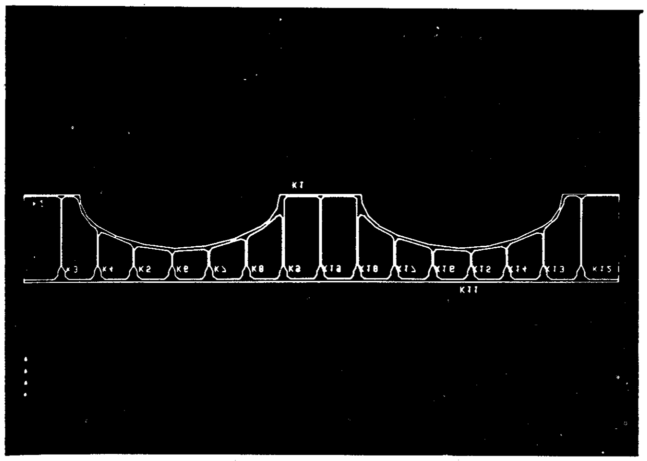
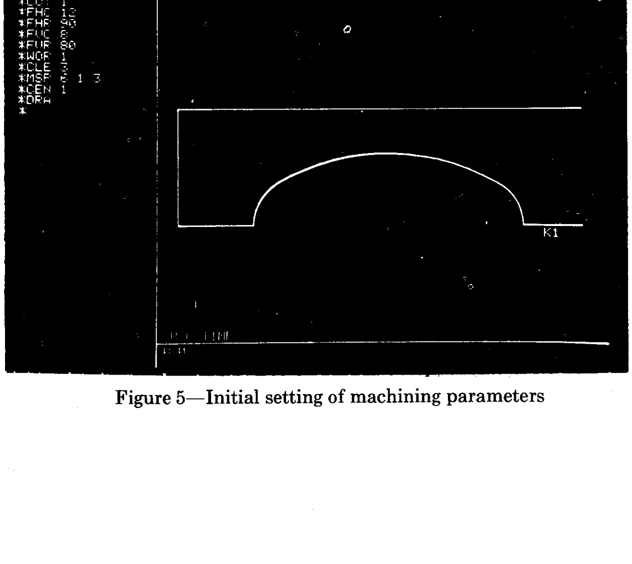


Figure 4—Checking geometry definition

Example of the use of GNC

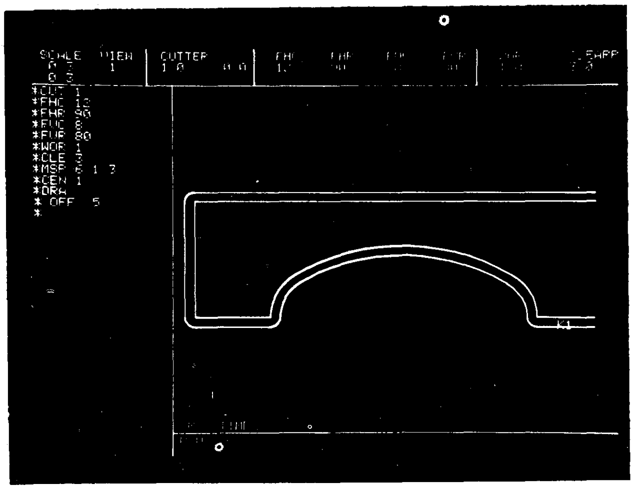
To demonstrate how GNC can be used a sequence of figures have been prepared showing the various stages to machine part of one of the components shown in Figure 1 from a solid block of aluminium. The component geometry was prepared using the KCURVES program, a detail of one of the pockets being shown in Figure 3. The geometry definition of the component is first checked by a drawing on the screen (Figure 4). After the selected cutter size, feed rates, height at which cutter will clear the metal and depth to which cutter must penetrate the metal have been specified (indicated by the row of numbers along the top of Figure 5) the setting point for the n/c machine is given (indicated by the "*" at the bottom left hand corner of Figure 5).

To remove the bulk of the unwanted metal, the centre of the cutter is controlled by movement of a cursor on the storage tube; this is called line milling. To prevent fouling of the finished profile, the cutter centre must remain a



```
SCALE   VIEW  CUTTER     FHC  FHR  FUC  FUR         CLEARP
 0.3     1    1.0  0.0   12                           3.0
 0.3
*CUT 1
*FHC 12
*FHR 90
*FUC 8
*FUR 80
*WOR 1
*CLE 3
*MSP 5 1 3
*CEN 1
*DRA
*
```

Figure 5—Initial setting of machining parameters



```
*CUT 1
*FHC 12
*FHR 90
*FUC 8
*FUR 80
*WOR 1
*CLE 3
*MSP 6 1 3
*CEN 1
*DRA
* OFF 5
*
```

Figure 6—Offset of component shown

distance at least the radius of cutter from the final profile. To ensure this, an offset has been drawn equal to the cutter radius as shown in Figure 6.

Two successive movements, using the line milling facility of the system, were then carried out as shown in Figures 7 and 8, the cursor being used to specify the end point of each straight line movement. This way of using the cursor means that an operator can easily use his machining skills to specify the best movements to remove unwanted metal. This can be particularly valuable if special care must be taken because there are relatively fragile items in the component such as thin webs.

The finished profile is obtained by one command where the cursor is used to indicate a start and finishing point as shown in Figure 9. The other two orthogonal views can be obtained as also can an isometric view of the entire tool path and component as shown in Figure 10.

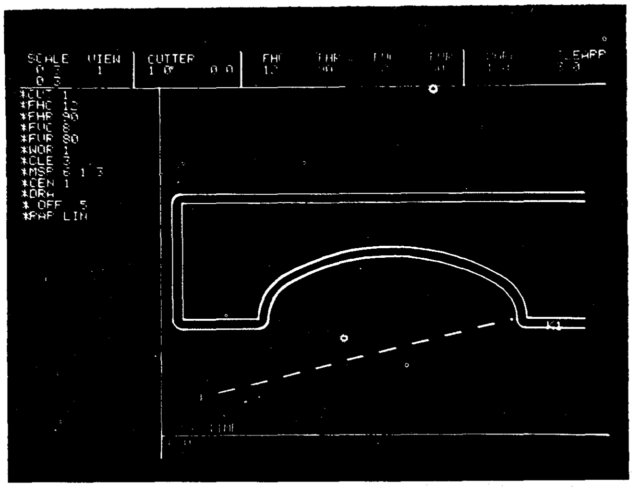
POLYSURF

Development of POLYSURF

Many components and objects found in various industries ranging from shipbuilding to bottle making are difficult to design because their shape is sculptured i.e., it has a free form which cannot easily be defined. Traditional methods of designing such objects are usually imprecise and may be restricted to the definition of a series of sections through the object as shown in Figure 11. When a component is manufactured from such a limited definition, the final detailed shape would be left until manufactured. Thus, if the component were to be moulded, the final shape would be determined by the pattern maker who possibly would make no reference to the designer but would rely on some "accepted code of practice" within that particular industry.

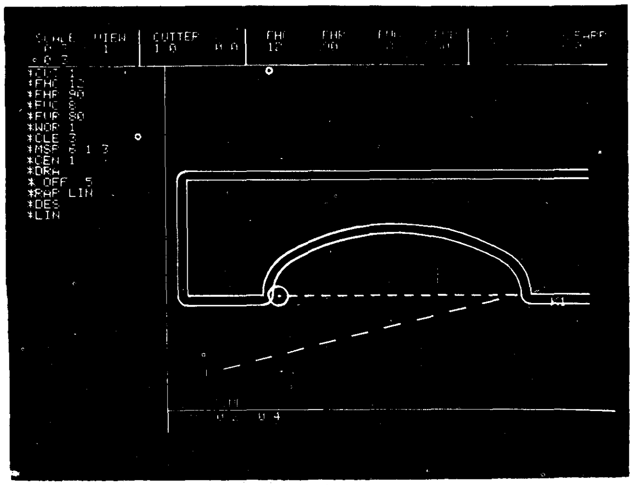
The fundamentals of computer techniques to overcome the difficulties of imprecise definition were developed as early as 1966.[4] Work at several places within the United Kingdom has been carried out since that time. Of these, the CAD Centre was able to draw directly on the research and development experiences of Cambridge University and BAC, Weybridge to specify the POLYSURF[5,6] system which (although still under development) can be used for the design and manufacture of a wide range of sculptured components. This research and development work showed that the inexact and incomplete definitions of components implied by traditional methods which simply define the geometry as a series of section lines, could be overcome by using mathematical equations to define the components' shapes. This solution had to be computer based because any practical mathematical definition needed a computer to carry out the necessary numerous calculations.

To develop a satisfactory system, and to be satisfactory it was essential that designers and production personnel from industry could readily use the system, two main



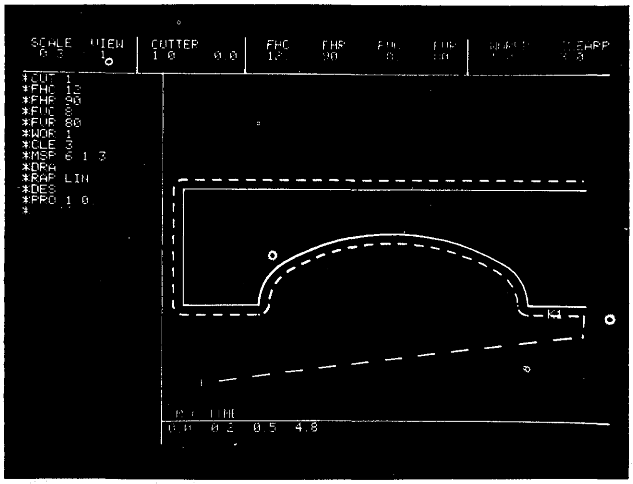
```
*CUT 1
*FHC 12
*FHR 90
*FUC 8
*FUR 80
*WOR 1
*CLE 3
*MSP 6 1 3
*CEN 1
*DRA
* OFF 5
*PHF LIN
```

Figure 7—First move using line milling option



```
*CUT 1
*FHC 12
*FHR 90
*FUC 8
*FUR 80
*WOR 1
*CLE 3
*MSP 6 1 3
*CEN 1
*DRA
* OFF 5
*PHF LIN
*DES
*LIN
```

Figure 8—Second move using line milling option



```
*CUT 1
*FHC 12
*FHR 90
*FUC 8
*FUR 80
*WOR 1
*CLE 3
*MSP 6 1 3
*DRA
*RAP LIN
*DES
*PRO 1 0
*
```

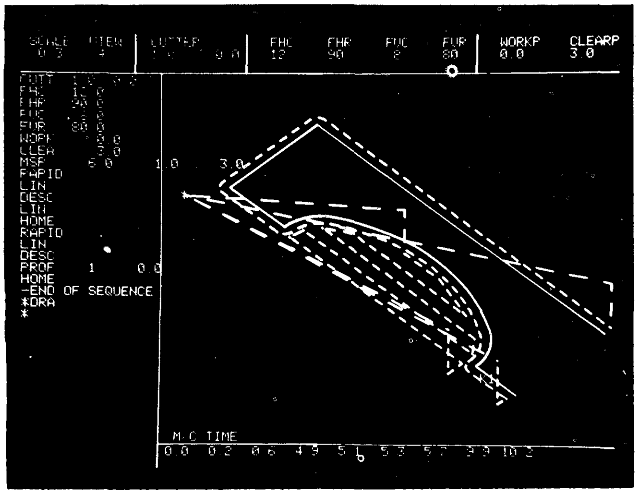
Figure 9—Cutting exact profile

types of problems had to be overcome, namely: the solution of the necessary mathematics of handling the equations used, and finding the means of using the mathematical solutions.

Both types of problem are important because poor means of using the mathematics would obscure the powerful tools that were provided and inadequate mathematical techniques would result in clumsy means of achieving satisfactory results.

During the time POLYSURF was being developed, various industrial organizations actively participated by working with Centre staff on specific design and manufacturing examples. The industries were widely based so that the list of components which were used as test pieces include:

- a shoe last
- ship hulls
- bottle moulds
- chair shells
- cab for high speed train
- turbine blades
- car dash panel
- jug mould
- sailplane fuselage.



```
CUTT 1.0 0.2
FHC 12.0
FHR 90.0
FUC 8.0
FUR 80.0
WORK 0.0
CLEA 3.0
MSP 6.0 1.0 3.0
RAPID
LIN
DESC
LIN
HOME
RAPID
LIN
DESC
PROF 1 0.0
HOME
-END OF SEQUENCE
*DRA
*
```

Figure 10—Isometric view of component and complete cutter path

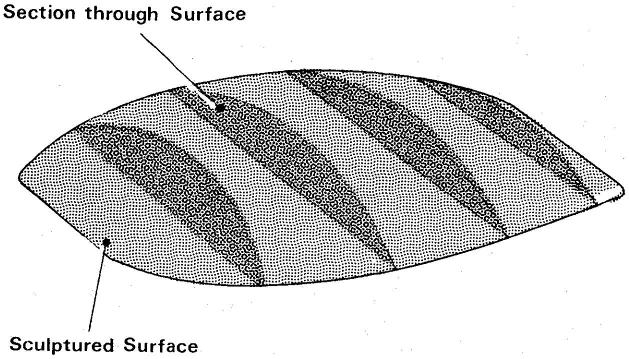


Figure 11—Sections through component with free form surface

In this way, pressure from demanding clients ensured that the POLYSURF system contained the necessary facilities which could be readily used. One consequence of using a computing means of defining components is that with the use of other computer methods which can generate half tone pictures of components (Figure 12), realistic pictures of the finished article can be obtained even before it is manufactured. This can be of great value during the initial marketing of the product as well as helping with the aesthetic appraisal of the design.

Facilities within POLYSURF

The POLYSURF system consists of three main modules which deal with the design of the component; drawing of the component, and production of tapes to control n/c machine tools.

This modular structure enables different sections of the program to be run independently so that the system runs

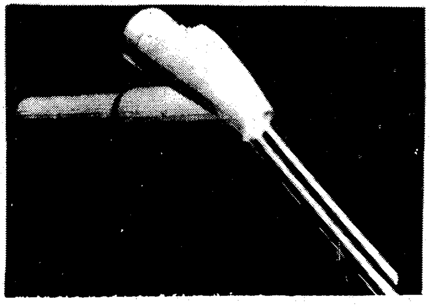


Figure 12—Computer generated half tone picture of instrument casing designed by POLYSURF

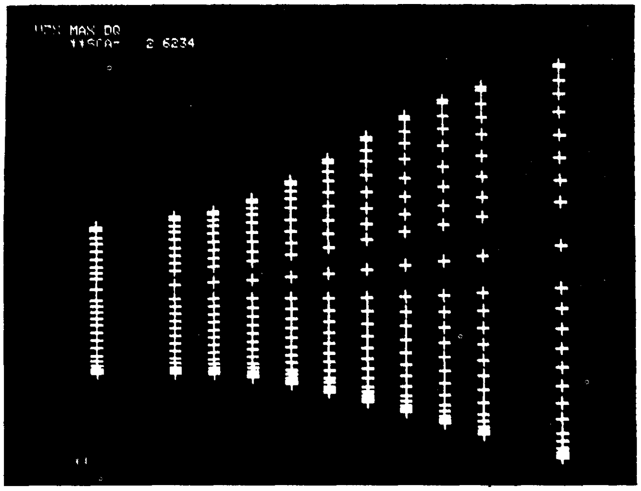


Figure 13—Distribution of coordinate data of blade

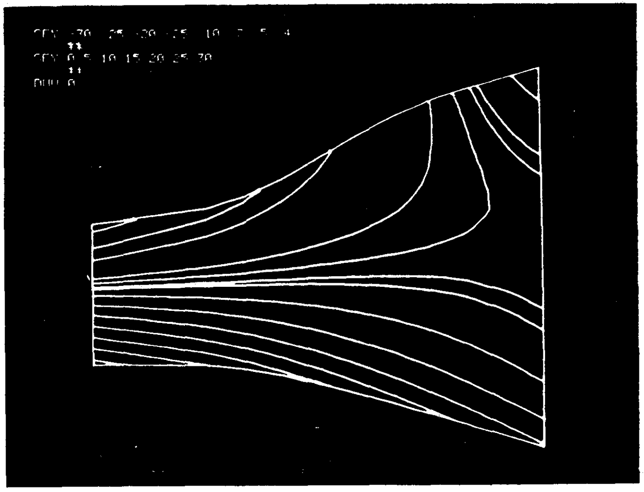


Figure 15—Sections through blade surface

with the minimum of overheads and can be used in a variety of specialized circumstances. The design module enables the component to be defined by the equations which not only apply to free form or sculptured surfaces but also to the more usual geometric forms such as planes, cylinders, cones and spheres. To assist the design, points can also be used as a means of checking whether or not the surface of the object passes through or near certain control positions. Geometric descriptions from the design stage can be stored for further manipulation by the design module to carry out any necessary refinements; the drawing module; the n/c tape preparation module, and other programs which are needed for specialist tasks like analysis or production of pictures by other computer programs.

The drawing module enables line drawings to be produced on a number of different graphic displays with the following options:

parts of, as well as, complete components to be shown
different scales or orientations to be chosen
different arrangements of views and perspective drawings to be selected
outline drawings to be produced
principal features to be included and
sectional drawings to be obtained in any chosen plane.

The n/c tape preparation module produces data suitable for standard post processors. Any area of the designed component can be cut by specifying the enclosing boundary as any combination of surfaces, planes, cones, cylinders and spheres. Additional options include differing cutting patterns and tool forms. The user must also specify the tolerance to which the surface must be cut.

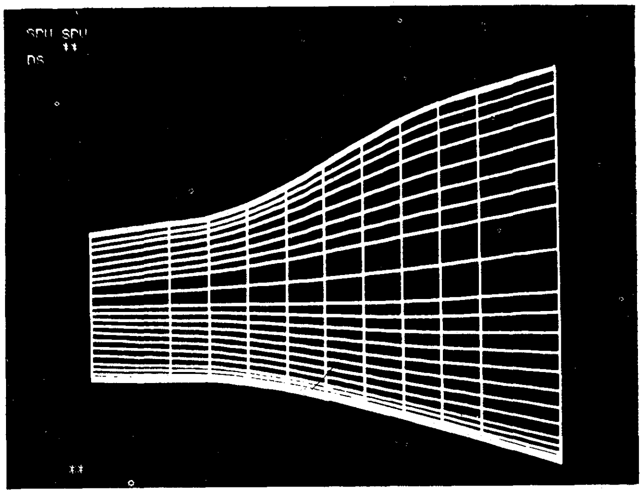


Figure 14—Smooth boundary lines on modelled blade

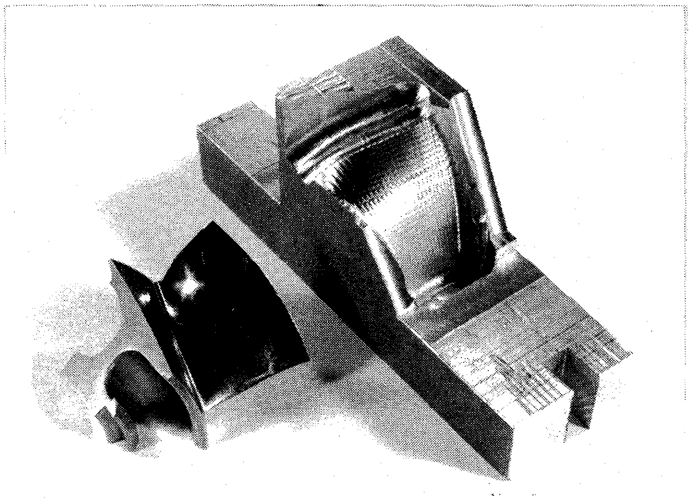


Figure 16—Finished blade and form machined using POLYSURF

Examples of the use of POLYSURF

An n/c machine tool supplier needed to demonstrate to a potential customer that the particular n/c machine was able to cut blades for compressors and turbines. The customer had already designed the blade and coordinate data was available at a number of points on both sides of the blade. Figure 13 shows the distribution of the points for one side of the blade. The POLYSURF system was used to fit a series of mathematical equations so that one equation applied to the area between each set of four points and there was continuity of position slope and curvature across the boundaries between adjacent areas. The resulting boundary lines between the areas can be seen in Figure 14 and sections at various depths are shown in Figure 15. The n/c tape preparation module was then used to prepare tapes to cut the blade surface between the leading and trailing edges which were defined in the original data; the cone which defined the root of the blade, and the cone which defined the top of the blade.

The blade surface was then machined giving the form shown in Figure 16 which also shows a finished blade.

ACKNOWLEDGMENTS

The author would like to take this opportunity of thanking those from industry, universities and the CAD Centre whose cooperation and ceaseless efforts have made it possible to describe in this paper the successful techniques which are forging new links between the design and manufacture of components.

REFERENCES

1. Davies, K. J., "GNC-A Graphical N/C Processor," *Proc. Prolamat 73,* North Holland Publishing Company, Amsterdam, 1973.
2. Anon, *GNC User Manual,* CAD Centre, Cambridge, 1974.
3. Anon, *KCURVES User Manual,* CAD Centre, Cambridge, 1974.
4. Coons, S. A., *Surfaces for Computer-Aided Design of Space Forms,* MIT, MAC-TR-41, 1966.
5. Flutter, A. G., *POLYSURF,* Ph.D Thesis, Cambridge University, 1974.
6. Anon, *POLYSURF User Manual,* CAD Centre, Cambridge, 1974.

Automatic program synthesis—From CAD to CAM*

by ROBERT T. CHIEN and TONY C. WOO
University of Illinois at Urbana-Champaign
Urbana, Illinois

INTRODUCTION

The technology gap between design and manufacturing has stimulated a number of research projects whose common objective is to bring design specification into some form of procedure that specifies the manufacturing sequence and tool path. Most notably, there are two related areas of study—sculptured surfaces and process planning. The former is concerned with representing non-analytic surfaces in parametric forms so that points on a surface can be generated from the parametric equations as cutting tool locations. The latter is concerned with grouping conventional machine parts and describing the part in special programming languages or representing them in codes to arrive at operation sequences, choice of machine tools, cutting data, and other information.

It is recognized that differential geometry is a powerful tool for representing sculptured surfaces. Coons' and Bezier's methods[1,2] have led to many developments and successful implementations of surface modeling systems with the use of computer graphics.[3,4] It is noted that there are logical operations in manufacturing, such as deciding the machining operations, or sequencing the operations, that cannot be represented in the framework of differential geometry alone.

On the other hand, studies in the area of process planning are just as successful. There are currently many systems in existence that accept special format inputs and generate work orders in batch or interactive modes.[5,6,7] Most of these systems deal with turned components by exploiting the symmetry of the part. Curiously enough, very little attention is given to the possibility of using designs done on computers as input to integrate the design and planning stages.

We see the problem of computer aided manufacturing as that of transforming geometric information into procedural information—from how something looks like into how it can be manufactured. We believe that there is enough information in a design to be processed automatically and to produce procedures for manufacturing purposes, if a person (a process planner, a part programmer, or a machinist) can do so with the same amount of information given.

In order to have a firm grasp of the problem, we have chosen three-dimensional machine parts as the problem domain. We feel there is sufficient structure in parts geometry and in numerical control machine tools for us to understand the problem and to develop the solutions.

To state our objective, we wish to associate the cavities in a given machine part design with the appropriate machining operations, sequence them, and expand the operations into numerical control programs.

DEVELOPMENT AND APPROACH

The basic issue addressed in this paper is that the shape of an object suggests a procedure. The handle of a tea kettle, a hammer, or a door knob suggest their functions in relation to the object they are attached to. Similarly, a familiar shape such as a hole in a machine part suggests drilling, a slot or a pocket milling. Our task is to attempt the construction of a manufacturing procedure from the shape information in geometry.

We understand that during the design stage, an operation such as "side mill roughing" can be associated with a particular cavity being designed. Consider the simple case of a hole. It may seem that attaching a label "drill" to a graphical representation would solve our problem. Not exactly. There are other key information that must be available before such a "drill" subroutine can be called. For example, is the hole all the way through? If not, how should the workpiece be set up, or from what direction should the hole be drilled? Should the hole be drilled together with other holes? Are there other holes of the same size? These are some of the specific questions normally asked by a part programmer when examining a design.

It would seem very convenient if a program can manipulate the representations of cavities at the level of bodies. As we learned in engineering graphics, a part can be visualized as a composition of simpler objects in an exploded view. This idea has been implemented in a computer aided design system by I. C. Braid.[8] Consequently, we assume an algebraic description of a machine part as obtained from Braid's system, i.e., in terms of a set of

* This work was supported under the Joint Services Electronics program (U.S. Army, U.S. Navy, U.S. Air Force) under contract DAAB-07-72-C-0259.

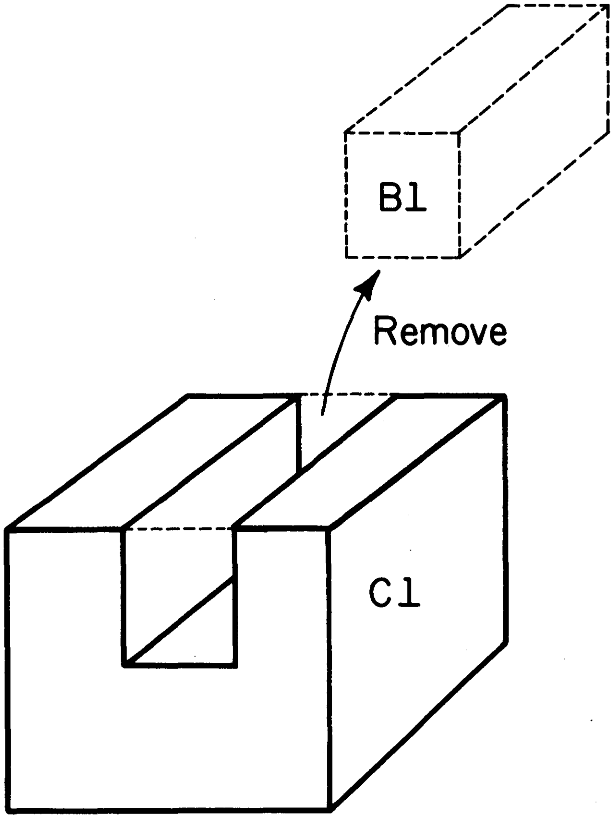


Figure 1—Create a slot by removing

primitive bodies such as cubes and cylinders, and relations such as adding and removing.

We see such a description as a very good way of telling to the computer how a machine part looks like, in a language of bodies. It is interesting to note, however, that there are many ways in which the same machine part can be described using the same set of primitives and operations. This is not particularly surprising because we humans can express an idea in natural language in just as many, if not more, ways using different components such as words, phrases, and clauses, and different arrangements of the components. Our problem here is in interpreting a body description of a machine in such a way that different descriptions of the same object should have the same interpretation.

To understand the problem of interpreting the geometry of a design, let us consider an example of a slot. A "slot" can be represented, as in Braid's system, by a negative body B1 initially transformed from a primitive cube (by scaling in the X direction, for example). If this negative body B1 is removed from a larger cube C1, a "slot" is formed. See Figure 1. The same object can be described by building up on both sides of the slot (yet to be formed). As in Figure 2, one could add two long blocks, B2 and B3, on a flat block B4.

We have just seen two of the many ways of syntactically representing a cavity in a computer. In order for the computer to carry out automatic operations such as using the right cutting tool, approaching the workpiece in the right direction, move the cutting tool a proper distance, it must first understand what it is going to cut. In other words, a program must first relate the geometry of a "slot" to information such as part surface, tool diameter, tool locations.

PARSING THE DESCRIPTION

A major concern in dealing with part descriptions is that it is a many-to-one geometry to machining concept mapping problem. In order to capture the intent of a design, we have constructed a grammar that transforms a design description into an internal representation in terms of machining operations. The grammar handles the six primitive bodies (cube, cylinder, fillet, sector, tetrahedron, and wedge) and commands (translate, rotate, scale, copy, negate, combine, and intersect) used in Braid's system.

Let us first analyze the structure of a design. We define an object (OBJ) as a concatenation of a modifier (MOD) and an object, where an object may be a primitive object or a modified object, and a modifier may be any combination of the three transformations—translation, rotation, and scaling.

OBJ=MOD OBJ
OBJ=primitive body/OBJ
MOD=translate/rotate/scale/MOD

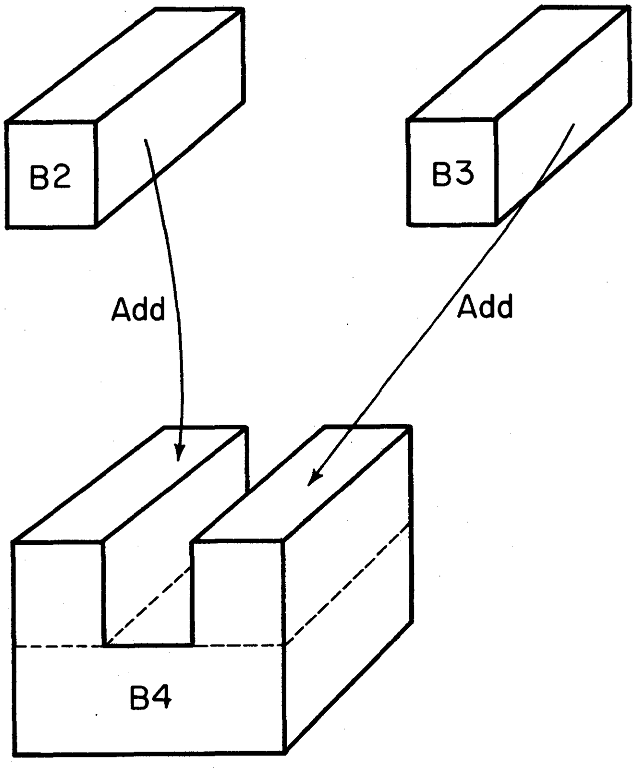


Figure 2—Create a slot by adding

An object can be operated on by copying, or negating it. We call the concatenation of an operation on a body an object group (OG).

OG=OP OBJ
OP=copy/negate

A machine part can be composed by joining several object groups together either by combining and merging the surfaces of the bodies involved, or by intersecting and creating new surfaces. A part is therefore a string of object groups joined together.

PART=OG JOIN OG
JOIN=combine/intersect

Having analyzed the structure of constructing a part from primitive bodies, we need to define the grammar for machining terminologies such as holes, slots, and pockets. Our grammar parses the design description by looking for possible constructs of such cavities. Since they are in the form of programs, they are best explained as decision procedures.

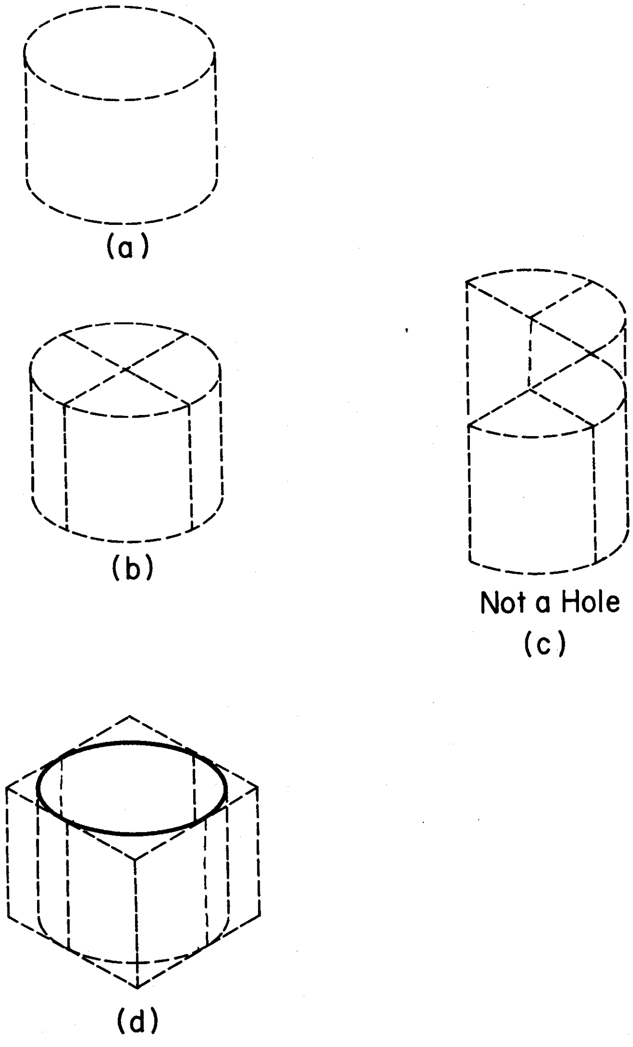


Figure 3—A hole procedure

We define a hole as a negative cylindrical object. A cylindrical object may be a modified primitive cylinder, e.g., a rotated and scaled cylinder. It could also be four sectors joined together forming a cylinder. See Figure 3b. Since joining four sectors together arbitrarily does not necessarily form a cylinder, as shown in Figure 3c, we look at the modifier for each sector involved to make sure that they are of the same size (the same scale modifier) first. We further note that rotating a symmetrical object with respect to a certain axis is a modulo operation. For instance, rotating a cube, scaled in the X-axis, 180 degrees around any of the three axes does not cause any change. Our program takes this into account and eliminates redundant transformations that may occur in the design. We also check the translation of each sector involved to make sure that the surfaces "mate" in the right manner. When these conditions are met, our program concludes that there are four bodies in the description of a machine part that forms a hole. A third possibility of a hole is the intersection of a cube with four negative fillets, i.e., taking away the corners of a cube. See Figure 3d. A similar procedure for checking each body involved is carried out.

Our program is context sensitive in that it not only checks the constituents in an algebraic expression of bodies and relations for a possible cavity, it also checks the relationship between the cavity and the body in which the cavity is supposed to reside. An example of this context sensitive aspect of our grammar is finding a rectangular open slot created in the manner shown in Figure 2. Our program first checks the scale, translate, and rotate modifiers of the two objects to make sure that there are indeed two parallel surfaces belonging to two different bodies flanking the "slot." It then concludes a cavity and checks for the size of the formed cavity with respect to the size of the two objects creating it. See Figure 4b. If the cavity is too wide, program perceives the two blocks as some sort of "walls" and does not treat the cavity as a slot. Our program also recognizes the importance of the relative sizes and orientations of the two bodies creating the slot and the body on which the slot is intended. If the two bodies are too small, the possibility of a slot is rejected. Similarly, if the two bodies are not oriented correctly, as in Figure 4c, the result is not interpreted as a slot.

MODEL OF NUMERICAL CONTROL MACHINE TOOL

After the cavities are interpreted, the results are passed onto another set of programs that produces an internal representation of the cavities in terms of cutting tool, tool diameter, approaching and cutting surfaces, and tool locations in space. Very often, our program returns more than one way of machining a particular cavity. This information is kept and saved for later processing.

At present, we have a model of a numerical control ma-

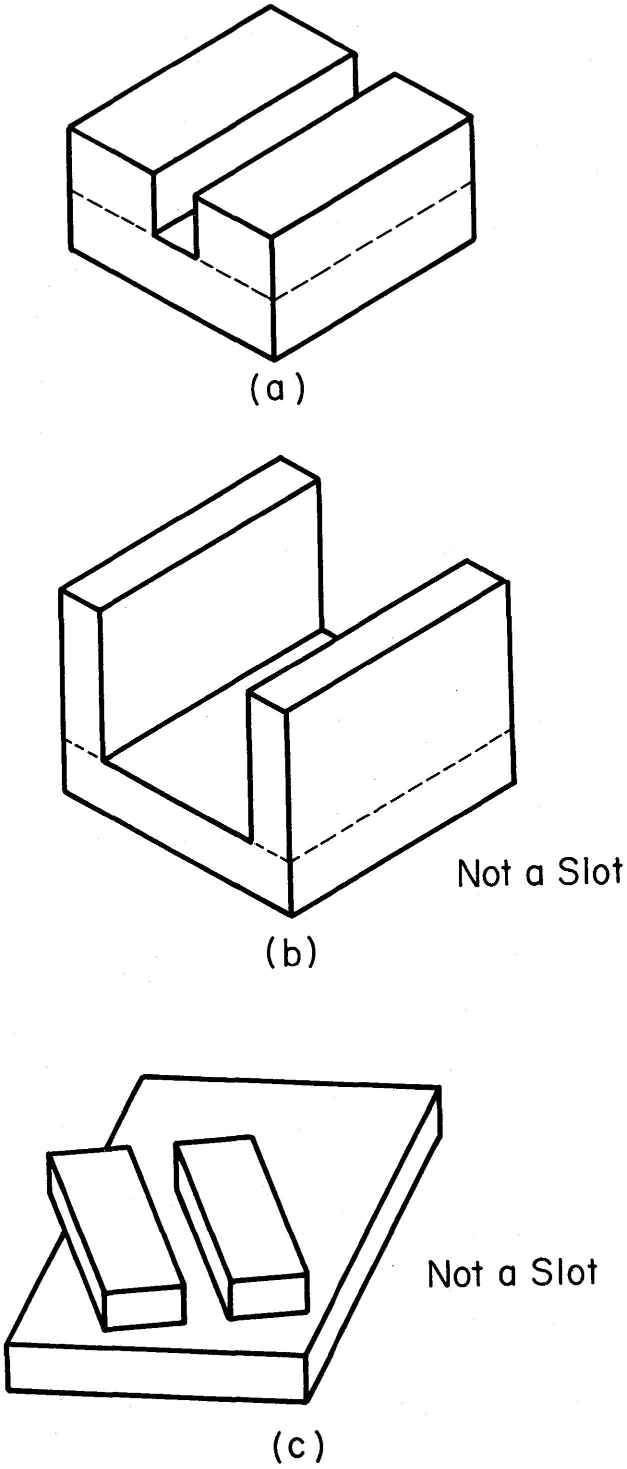


Figure 4—A slot procedure

chine tool which has the capability of 3-axis drilling and milling. Our model is built around the idea of a negative cylinder propagating in space. There are three such cylinders corresponding to the shapes of the most elementary cavities created by a drill, a side mill, and an end mill. Strictly speaking, the cylinders differ only in the bases, i.e., they are pointed, flat, and spherical, respectively.

The model is a collection of programs that simulates the cutting motions of the three kinds of tools. A slot, for example, is modeled as a two dimensional propagation of a flat-based cylinder moving in a straight line or following an arc. The result is a list of tool locations.

For bodies with complicated boundaries, such as a pocket with islands, a more general algorithm is needed to generate the tool paths. The basic ideas are partitioning the area enclosed and sequencing the partitions. This method is particularly useful in dealing with pockets with a concave boundary and with islands inside the pocket. Since the boundary is composed of line segments and arcs, local concavity points are first computed. They are next sorted according to their locations. The area enclosed is then partitioned into several regions with parallel line segments connecting the local concavities to the boundary. These regions are then traced and labeled. Since cutting is assumed in an increasing X, increasing Y manner, the regions form a lattice. They are partially ordered in the sense that a region cannot be cut unless the ones below it have all been cut. Lattice traversing algorithms have been developed for this purpose. The entire boundary including those for the islands is then digitized using scan-line conversion technique.[9] The digitized boundary is then offset and sorted in the increasing X, increasing Y order; the points provide a zig-zag tool path that runs between the boundary of the pocket and that of the islands.

In general, before these programs are called, more information on the cavity should be obtained. A special program for a specific kind of cavity is called to analyze the tool required. At present, there are ten sizes for each of the three types of cutting tools available. If, for instance, a hole is too large for drilling, an alternative machining method, milling in this case, is taken. In addition, the program checks what surfaces of the cavity in question are external, hence accessible to the cutting tool, and what surface the tool should reside on (the part surface in APT). If there is more than one possibility in which the tool could approach the cavity and does not penetrate other parts of the workpiece, they are reported to programs to be described in the next section.

GROUPING AND SEQUENCING

With the cutting tools selected, and the possible approach surfaces available, our program next sorts the cavities into groups. Basically, the program attempts to use the same cutter as much as possible without another set-up.

The algorithm proceeds by first grouping all cavities according to the plane on which their approach surfaces lies. This implies the number of set-ups. Very often, a cavity occurs in different groups because it may be cut from a number of directions. The multiple occurrence is first partially eliminated by considering the number of identical operations within a group. If there are less than three of the same kind, and if the group is not the last one in the list, the occurrences are deleted from the group. There may still be cavities occurring in more than one group after this operation. A semi-circular cut-out is an example. It can be interpreted as a hole and as a slot, thus it can be machined in more than one way provided the required cutting tools are available. The groups are next divided into subgroups according to the type of cutting tool needed. An elimination of multiple occurrence may happen if there are less than three cavities using the same kind of tool. If multiple occurrence still exists, drilling operation is given the preference to milling.

CODE SYNTHESIS

Our code synthesizer is quite simple. It is APT-like and has two kinds of information. If a list of points is given to it, the result is a series of statements of the form:

GOTO/Pi

If a list of line segments and arcs are supplied, it produces statements of the form:

GO*/Li%Li+1

where the modifier, *, is to be replaced by LFT or RGT, and % is to be replaced by TO, ON, or PAST, depending on the angular relationships between Li and the segment preceding it Li-1, and the one following it, Li+1. A computer graphics package is written that translates these statements into graphics commands, thus enabling one to visually examine the tool paths on a machine part.

CONCLUSION

We view our system as a pilot study of a totally integrated manufacturing automation system. Our objective, as we indicated earlier, is to create an environment in which design information can be utilized to produce manufacturing procedures directly and automatically.

We have shown that the problem of manufacturing is one of transforming graphical descriptions into program descriptions. We have divided the problem into two stages—obtaining from the many ways of describing a machine part in primitive bodies a kernel description of a machine part in terms of cavities, and interpreting the cavities in terms of the capabilities of a numerical control machine tool. It is hoped that our work provides the basis for bridging the automation gap between design and fabrication.

REFERENCES

1. Coons, S. A., *Surfaces for Computer-Aided Design of Space Forms*, MIT MAC-TR41, June, 1967.
2. Bezier, P., *Numerical Control—Mathematics and Application*, John Wiley Sons, 1970.
3. *POLYSURF User Manual*, Computer Aided Design Centre, Cambridge, England, 1974.
4. *SSX4 Sculptured Surfaces Project*, Computer Aided Manufacturing International, Arlington, Texas, 1974.
5. Bockholts, P., "TNO Miturn Programming System for Lathes," *Proceedings of PROLAMAT '73*, Budapest.
6. Hellstrom, P., "An Interactive System for Operations Planning for Turning on Centre Lathes," *Proceedings of PROLAMAT '73*, Budapest.
7. Sohlenius, G., "CAM-PRAUTO-DRILLING," *Proceedings of CAM-I Congress*, Hamilton, Ontario, Canada, May, 1974.
8. Braid, I. C., *Designing with Volumes*, Cantab Press, Cambridge, England, 1973.
9. Metzger, R. A., "Computer Generated Graphic Segments in a Raster Display," *Proceedings of SJCC*, 1969, pp. 161-172.

Automatic visual inspection*

by WESLEY E. SNYDER
University of Illinois
Urbana, Illinois

INTRODUCTION

The growing popularity of automatic inspection techniques was evidenced by the attendance at a conference on automatic inspection and quality control in Chicago last October.[1] At that conference, several reasons were presented to justify the development and implementation of automatic industrial quality control systems. These include: (1) releasing workers from tedious or repetitive tasks, (2) enabling precise and continuous inspection in inaccessible environments or under hazardous conditions, such as measuring the thickness of hot steel, (3) improving inspection system reliability, and (4) improving inspection system predictability.

Improving inspection techniques benefits a manufacturer in two ways. First, the finished product reliability is improved, and second, by inspecting during rather than after assembly, flaws can be caught earlier, resulting in rejection of a defective component rather than an expensive finished product.

One type of inspection which was touched on only briefly at the Chicago conference was automatic visual inspection. Such inspection techniques were considered for the most part a thing of the future. Automatic visual inspection techniques have been investigated to some extent, however, and this paper is a report on such systems.

For the purposes of this paper, it is convenient to restrict our definition of "visual" inspection from the broader "optical" inspection. Optical inspection could mean any sort of system using light. Many such applications are in use or under development, including especially the use of lasers. Visual inspection refers only to the processing of two-dimensional arrays of data in a computer's memory, where the numerical value of each array entry is a function of the light intensity at a corresponding point in a camera's image plane. Such images are easily obtained from television cameras, using analog-to-digital converters.

An example of an automatic visual inspection system is a system developed at the University of Illinois which uses a computer to perform visual inspection of hybrid circuits.

A primary motivation for developing this system was to examine certain programming theories and techniques. However, since those techniques are adequately described elsewhere,[2] we will not discuss them in this paper. Instead, we will describe the operation of the system and some of the "tools" which it used, such as color information and recursive region growing. We will then look toward the future of such visual inspection systems.

The system we will describe was developed on a PDP-10, a large, general-purpose computer. However, for industrial applications of such systems to be cost-effective, they must be implemented on minicomputers. Thus, in our discussion of the future of such visual inspection systems, we will describe some ways in which high speed hardware and high powered software can be used on minicomputers.

This paper is intended to provide the reader unacquainted with image processing some idea of the applicability of image processing to automatic inspection. This is not a complete state-of-the-art survey, in either automatic inspection or image processing, and those references included are usually chosen because they are easily readable and provide more extensive bibliographies themselves.

HYBRID CIRCUIT INSPECTION—AN EXAMPLE OF AUTOMATIC INSPECTION

Hybrid circuits are the middle ground between printed circuits and integrated circuits. In a printed circuit, only the conductors are printed onto a substrate, generally in the form of a metal film or "foil." The other components are individually mounted by pressing leads through holes in the board, and soldering these leads to the foil.

In an integrated circuit, all the components are an integral part of a single piece of silicon crystal. Resistors, transistors, diodes and some conductors are implemented by diffusing impurities into the wafer. Metal conductors may also be deposited on the surface of the crystal.

Construction of a hybrid circuit begins with a rigid, insulating substrate, generally some ceramic material. Resistors and conductors are painted on by using a silk screen painting process and paints, which, when dry, will have different resistivities.

Transistors, capacitors, and integrated circuits are then mounted and secured with solder or glue.

Finally, wires are attached which will make the connection to a frame.

* Research supported by the Joint Services Electronics Program (U.S. Army, U.S. Navy, U.S. Air Force) under contract DAAB-07-72-C-0259.

A major advantage of hybrid circuits is that they can be adjusted or tuned by changing the values of specific resistors. A popular way to accomplish this adjustment is by trimming the resistor's area down with a laser until the circuit performs in a specific way.

Visual inspection of hybrid circuits cannot be accomplished by simply checking for a point by point identification with a prescribed model because the manufacturing process introduces numerous irregularities which are not defects. For example, in the soldering process the solder rosin may flow in unpredictable ways, discoloring the substrate and producing extraneous edges. Furthermore, the laser trimming causes lines to be present on the surface of resistors, and these lines may vary greatly from one circuit to the next.

Several of the items in a hybrid circuit also exhibit a visual property commonly described as "shininess". That is, they are almost perfect reflectors of light. This has the annoying effect that a shiny region cannot be characterized in terms of either its characteristic shape, color, or intensity, as all will change with changes in the lighting or viewing angle.

The main characteristic of hybrid circuits which makes their analysis at all tractable is that although wires and straps and solder pads may obscure the view of components and may cast shadows, the circuit is primarily two-dimensional in nature.

In hybrid circuits factories, the visual inspection is generally performed by several inspectors peering through magnifying glasses or microscopes. Since the circuits can be tested electrically, by computer, for good or defective, yes or no, decisions, one major purpose of the human inspectors is quality control. They would like to determine not only whether a circuit is defective, but why it is defective so that corrective measures may be taken.

The inspectors check for missing, misplaced or misoriented parts, bad solder bonds, and anything else which looks wrong. In the "anything else" category come globs of glue or solder dropped on some component, cracked substrates, etc. While human inspectors may also use tweezers to check for mechanical continuity, this computerized system is restricted to detecting errors by purely visual inspection.

The inspection system

The inspection system consists of two major parts, a learning portion and an inspection portion.

The learning portion learns the characteristics of a particular hybrid circuit via a graphics display and a human teacher.

Once the learning portion completes its task, it generates a model of the circuit. That model is a computer program which, when compiled, becomes the inspection portion of the system. The inspection portion can then be run on other circuits of the same type to check them for defects.

Positioning of the circuit is accomplished via a simple jig which is positioned by "eyeballing." A pair of high intensity desk lamps provide lighting. Some typical messages from the system are:

```
DEFECTIVE RESISTOR AT 35 121
WIRE FROM 105 210 TO 36 240 IS MISSING
SUBSTRATE IMPROPERLY ORIENTED
NO DEFECTS DETECTED
```

TOOLS FOR VISUAL INSPECTION

Depending on the type of product being inspected, various algorithms can be used to derive significant characterizations or "features". Such features can be used for verification or recognition. The applications to recognition are surveyed well in a paper by Kanal[3] and in Duda and Hart's book.[4]

This section is a discussion of some of those tools. They include edge detection, region growing, and various intensity and color functions. This section does not cover all such tools. It is intended to provide only an indication of the types of operators available. Since the use of color has not received quite as much attention to date as the other types of tools, the problems involved in developing a suitable representation for color occupy a good portion of the section.

Fourier techniques

The two-dimensional Fast Fourier Transform provides a computationally efficient means for transforming between a spacial domain and a two-dimensional frequency domain. The transform of a well-defined spacial frequency will appear as a distinct feature in transform space. Fourier techniques are especially useful in inspecting products with highly repetitive patterns such as fabric. In such inspection, the presence of an undesirable frequency in transform space indicates a nonuniformity in the pattern and consequently a defect.

It is also possible to apply various filtering algorithms to the transform to reduce the effects of noise.

Fourier transforms of images can also be taken optically,[5] providing a decrease in computer loading.

Template matching

Template matching consists of comparing, usually by correlation techniques, a section of the image being processed with a stored "typical" image of a particular feature. Template matching may be used for feature location by rotating or translating a particular template over the image to maximize the correlation. It may also be used for image or feature recognition by trying a number of templates and choosing the one which maximizes the correlation.

If the location of the feature to be checked can be precisely ascertained and there is little noise (in the form of meaningless variations in the picture) simple template matching is a fast and efficient way to inspect.

Edge detection

An edge in a picture is a sharp change in intensity usually corresponding to an edge of the object being viewed. It often occurs that the exact location of particular edges is crucial to the inspection process. For example, in inspecting printed circuits, the edge of the foil must be properly located to avoid short circuits (see Ejiri et al.[6] and Finin[7] for other work on inspecting PC boards). There are a number of edge detection techniques available, surveyed by Duda and Hart[4] and Underwood and Aggarwal.[8] Edge detection is performed in the hybrid circuit inspection system by a sophisticated edge-detection and noise filtering algorithm developed by Hueckel.[9] This operator can detect edges of a prespecified strength with a prespecified confidence. It returns the equation of the newly found edge, and other information about the nature of the edge.

Region growing

There are many instances where the boundaries of a region are either ill defined or meaningless. For example, in the case of detection of wires, the glaring edges which the system attempts to locate are amorphous globs having only a roughly elliptical shape. For this reason, region growing is more useful than edge detection. The region growing algorithm uses simple recursive expansion of a region about a "seed". The criterion for whether a point is in the region is that its intensity exceeds a prespecified threshold, and it has at least one of its neighbors in the region. The region grower can return a list of the boundary points and some of the properties of the region, e.g., area and perimeter. Various approaches to region growing can be found in Brice and Fennama[10] and Yakimovsky and Feldman.[11]

The use of color

Color information can also be used. One approach is as follows: First, the three pictures which were taken using the color filters are corrected for the spectral response of the filters and the vidicon (this correction is done during the picture taking process). The color of a point is then initially defined as a three component vector, $<R,G,B>$ where R, G, and B are the intensities of the point as measured through the red, green and blue filters respectively. The magnitude of this vector is

$$\sqrt{R^2+G^2+B^2}$$

Division by this magnitude produces the "normalized color" of the point $<r,g,b>$. This triple, which is independent of the intensity of the point, could be used as the internal representation for color. Other systems[12] have used this representation. We did not because of the large memory requirements.

The basic problems with maintaining and using the three-color representation are the extensive core requirements, and the fact that it does not correspond neatly with any intuitive idea of "color" which the programmer or trainer may have. Thus it is desirable to represent the color of a point by a single number, rather than by three numbers.

To achieve this objective, the following approach is used in the hybrid circuit inspection system:

A color is more or less "saturated" depending on whether its white component is small or large. Thus, to remove the effect of saturation, the first step is to remove the white component.

Define cw(white component) $=$ min (r,g,b). Then, the "hue" is the vector $<r-cw,g-cw,b-cw>$. This is actually a two dimensional vector, since one of the components has been set to zero by subtraction.

Define T (the total intensity remaining after subtraction) $=$

$$(b-cw)+(g-cw)+(b-cw)$$

Now, the new, normalized hue components (one of which is zero) are

$$rn=(r-cw)/T$$
$$bn=(b-cw)/T$$
$$gn=(g-cw)/T$$

The simple thing to do at this point is to consider the spectrum as a linear graph, on which we have two components at two different frequencies, and compute the average of those two frequencies, weighted by their respective intensities. This, however, leads to a problem. Suppose the two components are of equal weight, red and blue. Then, if a simple weighted average of frequencies is computed, the result is a frequency in the green. People, however, do not see green in this circumstance, they see magenta.

This problem is resolved by considering the spectrum to be a circle instead of a straight line. On such a circle, red can be located at an angle of zero degrees, green at 120 degrees, and blue at 240 degrees. Thus, the average of red and blue comes out not in green, but somewhere between red and blue, as would be expected. A child describes magenta as "reddish blue", the same result the computer obtains using this approach.

This approach is roughly equivalent to considering a point in the inside of the color triangle and extracting its hue by projecting it out to the closest edge of the triangle along a line connecting the opposite (zero valued) vertex with the actual color point. The color triangle is described well by Cornsweet[13] along with many psychological and physiological aspects of color vision in people.

Representing the hue by the angle subtended on this circle gives a single number representation of hue, and thence of color.

But not quite, for hue is only meaningful for highly saturated colors, i.e., those colors with a relatively small white component. If an object is basically white, its hue is meaningless. Thus before using the hue as a test of color, one must first test the saturation to determine whether the object being viewed has any significant hue at all. Saturation is computed as follows:

Define s (saturation) $= 1 - (r+b+g-3^{*}cw)/(r+b+g)$

This function, saturation, is a maximum for a pure color or pair of colors, and is zero for white.

Whether hue or saturation is most meaningful depends on the type of object being viewed. In all cases, the hue cannot be reliably used as a test of color unless the saturation is high. Similarly, neither hue nor saturation is meaningful unless the intensity is fairly high. However, given the a-priori knowledge that the saturation should be high, one need only test one number, the hue, to determine whether the color of a region is correct.

Exactly how this representation is used depends on the characteristics of the object being inspected. The usual method is to identify the average hue of a region as a property of that region.

Other work on color has been studied by Land and McConn[14] and Tenenbaum *et al.*[15]

AUTOMATIC VISUAL INSPECTION USING MINICOMPUTERS

Whereas the last section was primarily a discussion of some software techniques for image processing, this section emphasizes hardware, for as soon as one attempts to apply the software techniques to real inspection problems at assembly line speeds, one is faced with the restrictions imposed by the type of imaging hardware being used. These restrictions are covered in more detail by Chien and Snyder.[16]

A tradeoff between speed, dynamic range, and resolution is essential when reading a TV picture into any storage medium. For example, it is impossible to read more than about 100 8 bit intensity points per TV line into a minicomputer (having 16 bit words) in one field time of a TV scan. Such resolution is not sufficient for many tasks. A PDP-10, on the other hand, with its 36 bit word can support more than twice this resolution. The longer word length of larger computers appears to be necessary to get both speed and resolution.

However, most industrial applications for computer vision are economically justifiable only if they employ a minicomputer. It is therefore desirable to consider how both the high resolution and speed can be achieved with a minicomputer-based system.

Means for using minicomputers for computer vision

It is initially tempting to solve this speed-resolution problem by simply buying fast memory. While such memories, generally made from solid-state components, are becoming less expensive and more readily available all the time, most minicomputer memory busses simply are not capable of supporting data rates as high as those we are considering and such high-speed memories are still relatively expensive.

Another way to approach the speed problem is by restricting the field of view. If the number of points to be examined in any one inspection is sufficiently small, then an image dissector[17] can be used to interrogate picture points in a direct-access manner, or one may use restricted scanning of a vidicon. Units which use both approaches are commercially available, complete with minicomputer interfaces.

Another way to solve this problem is to use a doubly ported memory on the minicomputer. In such a system, the memory actually consists of two memories and appropriate control logic to access them through different ports. Through the camera port, data enters both memories in parallel, making them appear as one memory which is twice as wide. Through the other port, the addressing is changed so that the two memories appear as interleaved minicomputer memory. Thus, by using such a doubly ported memory, it is possible to enter data at camera rates, retrieve it at minicomputer rates, and still make use of relatively inexpensive core memories.

While such a system is attractive in its simplicity, it suffers from not being generalizable. Having once established such a two-ported memory, one is likely to want to interface other devices to that memory. Other than a TV camera, some potential devices might be a raster scan monitor, a graphics system, or high speed image processing hardware such as a Fast Fourier transform taker.

For this reason, a more general approach can be taken. This approach is based upon a high speed general purpose bus vaguely similar to the UNIBUS in a DEC PDP-11. This high speed bus (HSB) is similar to UNIBUS in that data, address, and control functions are all carried on the same cable, and in that any device plugged into HSB can participate in data transfers to or from memory or any other device.

Such a HSB should be designed with two major considerations: to allow parallel operations among devices on the bus, including in particular, memory reads and writes, and to make the bus a general, expandable bus which can be extended to many devices.

CONCLUSION

The engineer about to design a visual inspection system must make several decisions. He must first consider the visual aspects of the objects he is inspecting. What are the significant features? Large areas of uniform intensity? Edges? Is color an appropriate parameter? How critical are positional tolerances? Such questions determine the type of inspection algorithms to be used.

How much resolution is needed to see fine detail? Is it necessary to inspect more than a very small region of the object at a time? What are the speed constraints? Must an inspection be completed in two seconds or two minutes? Such decisions, coupled with the speed of the processor and complexity of the inspection algorithms, may require random sampling or may allow inspection of every product. Such decisions also determine the selection of hardware.

This paper has attempted to introduce the power and flexibility of visual inspection. We have shown one example of a fairly flexible inspection system using rather high-level software techniques. We have also tried to provide the potential user of visual inspection systems with appropriate references to guide his decisions on both software and hardware.

REFERENCES

1. *Proceedings of the Conference on Automatic Inspection and Product Control,* IIT Research Institute, Illinois Institute of Technology, Chicago, October, 1974.
2. Snyder, W. E., *Automatic Visual Inspection of Hybrid Circuits,* Ph.D Thesis, University of Illinois, Urbana, Illinois, 1975.
3. Kanal, L., "Patterns in Pattern Recognition: 1968-1974" *IEEE Transactions on Information Theory.* Vol. IT-20, 6, November, 1974.
4. Duda, R. and P. Hart, *Pattern Classification and Scene Analysis,* John Wiley and Sons, New York, 1973.
5. Goodman, J. W., *Introduction to Fourier Optics,* McGraw-Hill, New York, 1968.
6. Ejiri, M., T. Uno, M. Mese, S. Ikeda, *A Process for Detecting Defects in Complicated Patterns,* Central Research Laboratory, Hitachi, Ltd. Kokubunji, Tokyo 185, Japan, 1973.
7. Finin, T., "Tracking Wires on Printed Circuit Boards," Working paper 52, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, October, 1973.
8. Underwood, S. and J. Aggarwal, *Methods of Edge Detection in Visual Scenes,* Information Systems Research Laboratory Technical Report No. 144., Electronics Research Center, University of Texas at Austin, 1973.
9. Hueckel, M., "An Operator Which Locates Edges in Digitized Pictures," *Journal of the Association of Computing Machinery,* October, 1973.
10. Brice, C. and C. Fennama, *Scene Analysis Using Regions,* Stanford Research Institute, Menlo Park, California, AI Tech note 17, 1970.
11. Yakimovsky Y. and J. Feldman, "A Semantics-Based Decision Theory Region Analyzer," *Proceedings of the Third International Joint Conference on Artificial Intelligence,* Stanford, California, 1973.
12. Yachida, M. and S. Tsuji, "Application of Color Information to Visual Perception," *Pattern Recognition,* 3, pp. 307-323, 1971.
13. Cornsweet, T., *Visual Perception,* New York, Academic Press, 1970.
14. Land, E. H. and J. J. McConn, "Lightness and Retinex Theory," *Journal of the Optical Society of America,* 61, pp. 1-11.
15. Tenenbaum, J. M., T. D. Garvey, S. Weyl, and H. C. Wolf, *An Interactive Facility for Scene Analysis Research,* Technical note 87, SRI Project 1187, Stanford Research Institute, Menlo Park, California, January 1974.
16. Chien, R. T. and W. E. Snyder, "Hardware for Visual Image Processing," *IEEE Transactions on Circuits and Systems,* March, 1975.
17. Horn, B., *The Image Dissector "eyes",* Project MAC, Massachusetts Institute of Technology, Vision Flash 16, Cambridge, Mass., 1971.

Automatic full-page formatting of technical primary journals

by STANLEY E. BAMMEL
Bammel Software Engineering
Columbus, Ohio

INTRODUCTION

Computerized photocomposition has been in use for several years at Chemical Abstracts Service (CAS) with excellent results for the production of high-volume abstract and index volumes (i.e., secondary publications) of a relatively simple type. The functions performed by the computer include selection of information from a data base, automatic organization of the material into the proper order, and formatting into composed units based on the data element types being processed. Simple hyphenation and justification are included, making use of arbitrary word breaks, but page make-up has required little more than breaking a continuous column of text into columns on a page.

Recently the composition techniques were extended in order to develop a system for photocomposing chemical journals (i.e., primary journals) in behalf of the Books & Journals Division of the American Chemical Society.

The overall approach to page composition is to break the problem into three parts:

1. Hyphenation and justification of text,
2. Layout of full-pages, and
3. Composition of figures, tables, and equations.

Full hyphenation and justification and full page make-up, including the placement of figures, tables, equations, etc., without human direction were required. The unique requirement was for automatic full page layout, and the design and implementation of the algorithm for part (2) is the subject of this paper.

At this time, figures, tables, and equations are produced manually and pasted into blank areas left in the composed text. Future extensions to the journal composition system will be aimed at computerized photocomposition of these non-text elements also, reducing manual operations to as few as possible. The first implementation produces INORGANIC CHEMISTRY.

GENERAL PAGE MAKE-UP RULES

Based on longstanding manual practice, the page make-up rules for INORGANIC CHEMISTRY are highly stylized but are not unusual for a technical journal. Text is placed two columns per page except for the heading of each article which is full-page width. Page depth is the same in all cases, and no blank area is permitted except at the end of an article when there is not enough room on the same page to begin the next article. Columns must be flush at the beginning and end of each article, at the top and bottom of each page, and below or above two-column-wide graphics. Text may not be broken in certain places; for example, a sub-heading and at least one line of text following must appear in the same column. Footnotes are placed at the end of each article. No non-manuscript material, such as advertising, appears in the main body of the magazine.

Minor graphics include equations and miscellaneous artwork which are closely associated with the text and are designated as either "read-in" or "as-soon-as-possible." The former must always appear at a specific point in the text with no opportunity for adjustment, but the latter may, if necessary, be delayed a small distance after they are referred to.

Major graphics are generally larger than minor graphics and have much more freedom of placement. Therefore they are treated very differently by the algorithm. There are several classes of them; for example, tables, figures, and charts are treated almost identically. Within each class, the author assigns a sequential identification number that determines the order in which they must appear in the article. Between classes there is no ordering; for example, Figure 5 could well appear before Table 2, but never Table 3 before Table 2. A given major graphic may be referred to in the text in several places, but only the first instance (the callout) is taken into account in placing the graphic. Although it is desirable for a major graphic to appear near its callout, it may be forced to appear pages away, usually because other large graphics must be placed first, just as if the layout had been prepared manually.

Major and minor graphics are always assigned to rectangular areas which are an integral number of columns wide. Graphics are never split between pages except for certain major graphics which may be larger than a page. The dimensions of graphics are fixed at the time of input and cannot be adjusted by the page make-up program.

Above and below each graphic and between certain text blocks appears vertical spacing which may be varied within a certain range in order to obtain correct vertical alignment of the columns. Occasionally when the normal range is not sufficient, extra spacing is inserted between paragraphs and, rarely, between lines.

Other detailed page make-up rules will be mentioned in context as appropriate in the following description of the algorithm.

THE PAGE MAKE-UP ALGORITHM

As a preliminary step, all hyphenation and justification is performed by the computer before entering the page make-up algorithm. The latter then receives as input only an "abstract description" of the text which includes such data as the type of text (e.g., main text, subheading, footnote, etc.) and the number of lines in each block of text. (A block of text is one or more lines but never more than a paragraph.) Similarly for a graphic, the only entities which are input are the size, the type, and the location of the callout.

It should be obvious that the total number of possible "acceptable" ways of formatting a given article may be very large. When layout is done manually, the selection of the layout is based on experience and skill in adjusting graphics and text into a form judged best for the specific article at hand. Although it would be technically possible to compute and evaluate all possibilities according to a set of placement quality criteria in order to choose the "best" one, it would be expensive. Furthermore, such brute force is not necessary since an efficient and effective heuristic procedure can be devised. The key to this or any other heuristic procedure is at each step to first select the most promising cases for further exploration and development rather than waste time on cases which are likely to prove unsuitable. In the algorithm developed at CAS, articles are formatted one at a time in the sequence in which they are to appear. Since the starting point of each successive article is dependent on the ending point of the previous article, the algorithm actually performs placement for an entire issue of a journal. For each article the algorithm falls naturally into three hierarchical phases, each invoking the next and all phases being performed for one page before advancing to the next page. The phases are as follows:

Phase I: Selection of the set of major graphics to go on a page,
Phase II: Placement of the selected major graphics on the page, and
Phase III: Placement of text and minor graphics in the remaining unoccupied area.

The overall algorithm is a backtrack procedure where each phase indicates success or failure to the phase which invoked it. For example, if Phase II indicates success, then Phase I proceeds to the next page. Otherwise, Phase I selects the next best set of major graphics for the page in question and invokes Phase II again. Should Phase I in such manner exhaust all reasonable possibilities for a given page, then it backtracks to the previous page (or as far back as necessary in the article) to select a set of major graphics which has not been tried. In a similar manner Phase II arranges on the page the set of major graphics selected by Phase I and then invokes Phase III. If Phase III succeeds, then Phase II indicates success to Phase I. However, each time Phase III indicates failure, Phase II rearranges the placement of the graphics on the page and invokes Phase III again. This can continue until there are no more distinct arrangements, in which case Phase II indicates failure to Phase I. The entire algorithm fails when all reasonable possibilities for the first page are exhausted. Experience to date indicates that this occurs rarely and when it does, it is generally for the same reasons that manual page make-up fails, for example, running out of text before placement of all graphics. The remedy is the same also—modify the article and reformat it.

Phase I—Selection of graphics for a page

Phase I is a tree search which selects sets of major graphics (referred to as "graphics" when clear from context) to be assigned to pages. Conceptually, the first page of the article is the first level below the root of the search tree, and each successive page is then one level lower. Each node represents a set of major graphics which is a candidate for assignment to the page corresponding to the level of the node.

Each time Phase I advances to a new page of an article it computes and evaluates all possible sets of major graphics and then discards (i.e., "prunes" from the search tree) unpromising cases. The graphics available for placement must be chosen from those not already placed on previous pages. Simple size checks and the ordering restrictions usually rule out many of the sets *a priori*.

From the reader's standpoint, the most important factor for convenience in reading is distance from the callout to the major graphic itself, and, other factors being equal, it is considerably more desirable for the callout to precede the graphic rather than to follow it. If the callout cannot appear on the same page as the graphic, then the next best choice is to place the callout on a facing page.

The most important factor, from the standpoint of achieving a feasible solution, is placing the major graphics as early as possible in the article. This is especially true with the large graphics which are naturally more difficult to handle. More of the text will thus be displaced towards the end of the article. This leaves more flexibility for rearrangement later within the article and makes it less likely that the algorithm will fail later, causing it to backtrack. This is a sort of global optimization strategy.

Therefore, the quality rating for a set is primarily based on these two considerations and is the sum of the quality ratings of the individual graphics in the context of the set. (The quality rating of a graphic cannot be computed independently of the set since the composition of the set may influence the location of the callout of the graphic.) The quality rating for an individual graphic is computed as follows such that the smaller the numerical value, the better the quality:

1. If it is estimated that the callout will appear on a previous page, then

quality = −(area of graphic) − (area of text from the callout to the start of the current page)
The quality will thus be represented by a negative value.

2. If it is estimated that the callout will fall on the current page, then
quality = −(area of graphic)
A negative value again results.
3. If it is estimated that the callout will fall on a subsequent page, then
quality = 2 × (area of text from the start of the current page forward to the callout)
The quality in this case will be a positive value.

There are also secondary considerations, for example, if the callout should appear on the following facing page, a modest bonus is substracted from the quality rating.

This simple scheme has given very good results, and there are other criteria, not currently implemented, which could fit in very naturally. For example, if it is desired that certain graphics appear together on the same page, then if they all occur in a set, a bonus (probably large) could be subtracted from the quality rating of the set; if not, a penalty could be added. This way the desired result will be strongly favored, but if it is impossible, page make-up is not prevented from proceeding normally.

Phase II—Placement of graphics on a page

Given a set of major graphics chosen by Phase I for a particular page, Phase II computes their placement on the page and indicates success or failure to Phase I.

For placement and ordering purposes, two-column-wide graphics are counted in column one and are all placed together at the top and/or bottom in order not to break up text in the middle. One-column graphics then go at the tops of the columns directly below any two-column graphics at the top of the page. A further restriction is that in a column, graphics must be contiguous (as well as ordered) with other graphics of the same class. For example, consider the following sequences in a column:

Valid: Figure 2, Table 1, Table 2,
Valid: Table 1, Table 2, Figure 2, and
Invalid: Table 1, Figure 2, Table 2.

The set of graphics to be placed in column one is selected from the set of graphics assigned to the page in the same general way that the set of graphics for the page was selected from all the graphics remaining to be placed. That is, all possibilities are computed, illegal and impossible configurations discarded, and those remaining are given a quality rating so the best one(s) may be tried first. Selecting the set for column one then determines the set for column two.

Legality checking includes size and sequence checking. For example, the sequence: Figure 1 (two column wide), Figure 2 (one column), Table 2 (two columns), Table 3 (one column) is illegal for column one since it is not possible to place all two-column graphics at the top and bottom and still have graphics ordered and contiguous within classes. Another legality check is that all two-column graphics must be assigned to column one.

A major factor in computing the quality is, as noted before, distance from the location of a graphic to its callout in the text. The quality factor for each one-column graphic is computed by estimating the location of the callout so that if it appears in a later column, an appropriate negative value is assigned, otherwise a positive value. Since there is no choice for two-column graphics, they need not be rated.

Phase III—Placement of the text

After major graphics have been formatted, the text and minor graphics are placed in the unoccupied area by an algorithm which is global to an entire article. Except for two factors, the algorithm can be logically viewed as being performed after Phases I and II have found a complete major-graphic placement solution for the entire article. However, in actuality, Phase III proceeds page-for-page in step with the first two phases. This is because of two factors: (1) Phases I and II use the locations or estimated locations of callouts, which are dependent on placement of the text, in order to evaluate the qualities of graphic sets and (2) if there are no feasible ways to place the text, the algorithm backs up without Phases I and II having wasted time on subsequent pages.

Due to the allowed variability of the spacing, it is usually possible to break a column of text in several places while satisfying the rule that columns must be flush at top and bottom. When more than one breakpoint from the previous column is considered, there are even more possibilities. A list of all such breakpoints is computed, and each breakpoint is evaluated by criteria such as closeness of spacing to optimum values and absence of widows. (A "widow" is generally a short line at the top of a column.) The global quality rating of each new breakpoint is the sum of the squares of the qualities of all previous breakpoints which occurred in the "path" to arrive at the new breakpoint. Frequently the same breakpoint will result via different paths. In such a case, only the path with the best global quality is retained. Then the list is ordered on the global quality and for each column a limited number (currently five) of the best are kept. The reason for using the sum of the squares is to rate a number of small deficiencies better than one large one.

Balancing of columns at the end of an article is achieved by first estimating from the area of the remaining text and spacing ranges the maximum and minimum possible height of the balanced columns. This estimated range is enlarged by a small safety factor, and then all possibilities are tried starting from each of the breakpoints retained from the end of the previous page. Quality is computed essentially as before. The overall best text placement for the article is then the solution with the best global quality rating.

An example

In order to simplify the presentation of the example, it will be assumed that Phase III operates column by column rather than global to the entire article. This does not alter the functions of Phases I and II. Other simplifications for illustrative purposes are to assume spacing is fixed and incorporated into the stated dimensions of the graphics and to evaluate graphic sets (in Phase I) by only the two primary considerations. It should also be noted that this example is somewhat atypical in that most often an article is formatted without any backtracking.

Figure 1 shows the pages in the order in which they are processed and is approximately drawn to scale. Text is represented by dashed lines, graphics by rectangles, and callouts are indicated in the text. The current text area (TA) is found at the upper left hand of each page (in column inches) from the beginning of the article to the beginning of the respective page. (Note that the area of minor graphics is included in the text area.) The height of the printed area of each page is 1Ø inches. The dimensions of the graphics and location of callouts is given in Table I. Assume that footnotes start at 27 column-inches and that the total length of the text is 3Ø column-inches.

TABLE I—Dimensions of Graphics and Locations of Their Callouts

	Height (inches)	Width (no. of columns)	Area (column-inches)	Callout Location (column-inches of text from beginning of article)
Table 1	3	1	3	16
Figure 1	2	2	4	18
Figure 2	5	1	5	21
Table 2	7	2	14	26
Equation 1	2	1	2	24

The following commentary follows each page as it is processed. (To derive the quality rating scheme see the description in Phase I and Table I.)

A. Due to the ordering restrictions and the fact that two-column-wide graphics may not appear on the first page, the only possibilities for page 1 are:

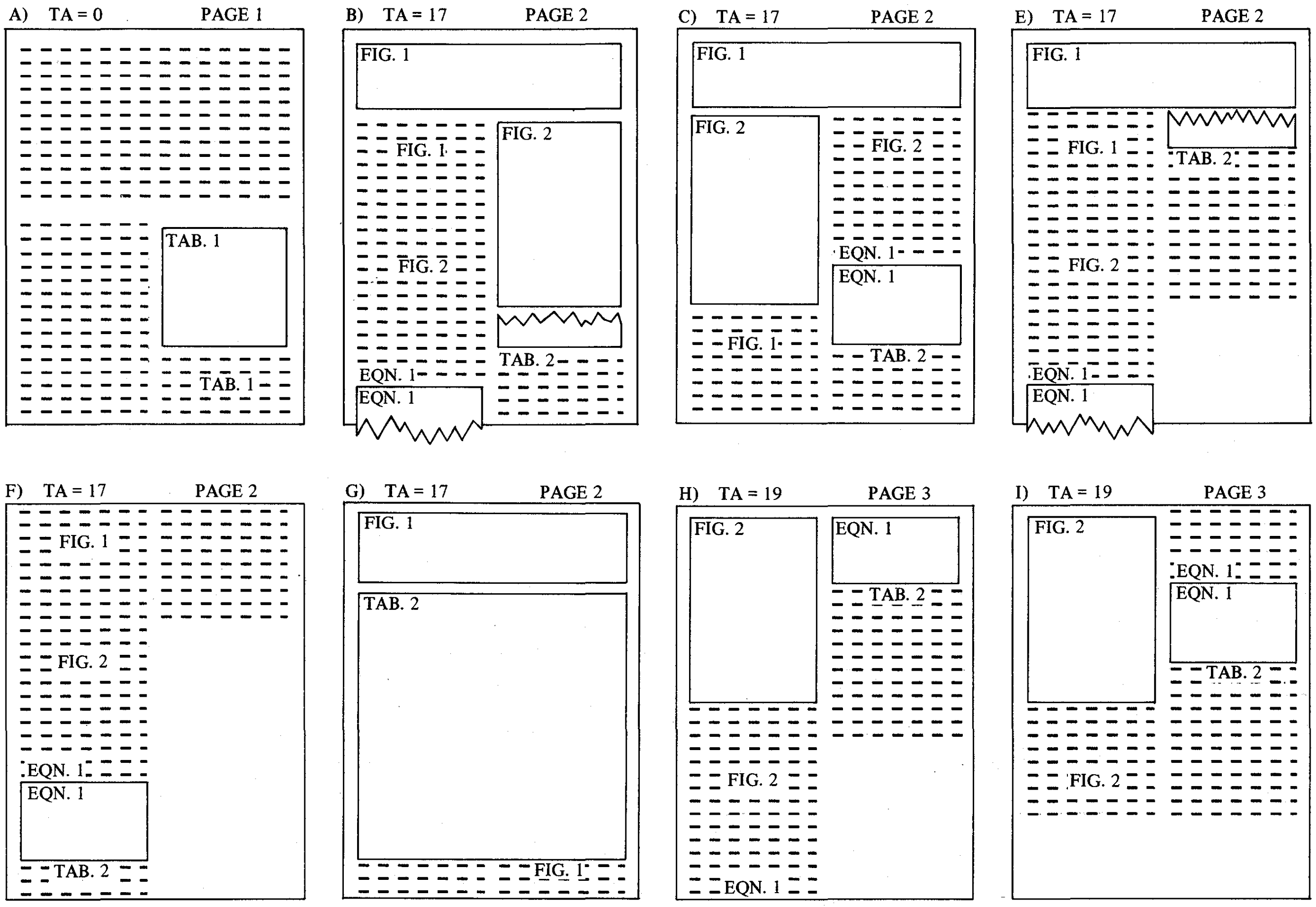


Figure 1

	Quality Rating
(a) Tab. 1	−3
(b) no major graphics	Ø

Possibility (a) is tried and found acceptable.

B. The possibilities for page 2 now are:

Quality Rating for Each Graphic in Parenthesis	Quality Rating (sum of individual ratings)
(a) Fig. 1(−4), Fig. 2(−5)	−9
(b) Fig. 1(−4)	−4
(c) no major graphics	Ø
(d) Fig. 1(−4), Tab. 2(18)	14
(e) Tab. 2(18)	18

Note that the set: Fig. 1, Fig. 2, and Tab. 2 is not considered because it is too large to fit on the page. Possibility (a) is tried but is not found acceptable because equation 1 falls across a column boundary; therefore, Phase III fails.

C. Phase II rearranges the graphics and this time Phase III succeeds.

D. The possibilities for page 3 are now:

	Quality Rating
(a) Tab. 2	−16
(b) no major graphics	Ø

Possibility (a) is tried first but Phase III fails because the rules do not permit a major graphic to appear on a page with only footnotes. Possibility (b) fails also because the amount of text is exhausted before all graphics are placed. (Not illustrated)

E. The next possibility [(b) from step B] for page 2 is tried but fails in Phases II and III because the read-in equation falls across the column break and there is no way to rearrange Figure 1.

F. The next possibility [(c) from step B] fails because the amount of text is exhausted before all graphics are placed.

G. The next possibility [(d) from step B] succeeds.

H. The possibilities for page 3 now are:

	Quality Rating
(a) Fig. 2	−5
(b) no major graphics	Ø

Possibility (a) succeeds.

I. When the end of the text is detected and all graphics have been placed, Phase III balances the column.

IMPLEMENTATION AND RESULTS

The page-makeup algorithm was implemented in PL/1 using structured programming techniques with design, coding, and debugging being accomplished in approximately one man-year. Except for the final stages, testing was greatly facilitated by printing on the line printer a mockup of the page layout in a format similar to that of Figure 1.

As noted previously, the algorithm more often than not formats an article without any backtracking, as indicated by the fact that less than 25 percent of the pages formatted in intermediate stages of the algorithm are rejected before arriving at the successfully formatted issue. Utilization of the PL/1 Optimizing Compiler on an IBM 37Ø/168 computer under operating system VS2, Release 1.6, with a test driver a 16ØK-byte region is required and run-time is approximately 0.5 seconds per page. Use of the production routines necessary for accessing the data base raise these figures to about 21ØK bytes and 0.8 seconds per page. Final output is produced on an Autologic APS-4 photocomposition system.

(The figures in the above two paragraphs do not include steps preliminary to and including hyphenation and justification, nor do they include operation of the Photocomposer.)

ACKNOWLEDGMENT

The contribution of Thomas J. Funk is gratefully acknowledged, in particular, for formalizing the page makeup rules and many helpful suggestions on the design of the algorithm. For further information concerning the work described in this paper contact the Director of Research and Development, Chemical Abstracts Service, Columbus, Ohio 43210.

A simple technique for controlled on-line system stimulation

by THOMAS E. BELL
TRW Systems
Redondo Beach, California

and

JO ANN LOCKETT
The RAND Corporation
Santa Monica, California

INTRODUCTION

Most performance analysis tools are developed with the objective of providing data in a general way, using flexible techniques, with confidence that analysts will subsequently employ them in cost/effective procedures. The inadequacy of this approach is reflected by the frequency of questions about which data to collect and how to use them. The problem is the initial emphasis on tools as an end in themselves rather than on analysis techniques.[1]

This paper describes research into analysis techniques for tuning on-line systems and the development of a hardware tool to facilitate analyses using these techniques. Although the potential for wider application of the tool is recognized, this was secondary to developing improved metrics, researching new methods for determining on-line system improvement, and understanding the performance relationships between on-line and batch systems.

Tuning an on-line system often requires that loads, either natural or artificial, be input to measure system response. Natural loads have been found deficient for some purposes because their detailed characteristics affect performance dramatically. Since precise replication of a natural loading situation cannot be attained, comparisons of system responses under different situations is nearly impossible. In contrast, a totally artificial load may have only limited applicability because it constitutes a significantly lower level of demand than a normal, natural load unless expensive techniques are employed. In addition, an artificial load is seldom like the natural load. This paper describes an alternative which combines the advantages of both natural and artificial loading.

LOADING ON-LINE SYSTEMS

An artificial load does not necessarily replicate real user activity. Therefore, work done at the National Bureau of Standards[2] recorded the actual inputs from real users in order to subsequently submit them to the computer under experimental conditions. In this way the reality of interactions was maintained while the script remained constant to be input to a computer under alternative hardware or software configurations. However, everything was recorded, making analysis difficult under most conditions.

A similar hardware device has been developed at the Mitre Corporation.[3] It produces simulated inputs from a number of terminals and records all interactions. However, the work did not include an investigation of metrics; thus the Line Load Emulator records all data of possible interest. Similar devices are typically used by computer vendors in developing on-line systems and in debugging them.

Software techniques are also employed in causing a computer system to respond as though real users are present. Some of these are university research tools (e.g., Stanford[4]) while others are commercial products (e.g., Tesdata Load Generator[5]). Software tools are machine-specific, but do not require expensive, special purpose hardware for operation.

Two advantages of these load generation tools are their internal data collection facilities and their ability to operate in the absence of the real users for the system under study. They can, therefore, be used in computer acquisition studies when little other information is available. They also have a series of limitations which seriously reduce their applicability in tuning studies, primarily:.

- If hardware, they are expensive to acquire.
- If software, they are usable on only one type of system, require significant operating system expertise in most cases, and may be expensive to acquire.
- They usually disrupt operations. Hardware tools require interfacing to some central part of the computer system. Software tools require memory space on the system and may degrade performance independently of the load being added to the system.
- They require considerable effort to design a script. Even minor script decisions may unexpectedly produce non-typical results.

DEVELOPING AN ANALYSIS APPROACH

The impetus for our research was an investigation of the effects of removing equipment from the Rand computer system on two on-line systems.[6] Since the purpose of the evaluation was to ensure that users not be adversely affected by the proposed equipment modification, the responsiveness of the on-line systems was investigated. This posed difficulties since no data collection tools were readily available to evaluate these systems. Definition and collection of appropriate response times for the system which used interactive vector graphics terminals appeared to be quite difficult. Since this system was receiving minimal, noncritical use, we were able to use a restricted set of techniques employing judgments of human subjects.

Evaluation of the other, more popular system provided an opportunity to explore new techniques of on-line measurement with the aid of a test instrument* which had been designed and employed at Rand for development and maintenance of the Videographics system.[7] This evaluation emphasized repeatedly determining the time required to process a fixed number of identical on-line requests. Several requests (using different types of system facilities) were employed both before and after the change. The load on the system was totally natural except for the slight incremental loading due to the artificial load introduced by the measurement. This basic approach guaranteed us a realistic load (the advantage of natural loading) and also enabled us to exercise some control (the advantage of artificial loading).

The basic approach was to employ single commands rather than complex scripts and to obtain multiple samples of performance in the natural loading environment so that the effects of variability could be determined. Unfortunately, applying this approach even for further development, suffered from three deficiencies:

1. Individual commands experienced extreme variability; even a few trivial responses with aberrantly long response times can make the system unacceptable to users, but the average of a large number of responses does not reflect such extremes. Timing of individual commands is therefore necessary.
2. The device that inputs the commands repeatedly would operate only with the Videographics System.
3. The response to a command is often complex with a few characters being displayed immediately to inform humans that the machine recognizes their request. Therefore, the definition of the end of an interaction is more complex than simply "the first bit of the response message".

Continued research into this approach required that a device be developed and built to eliminate these problems so that investigations could be performed on a variety of machines. The investigations needed to be made with individual commands using carefully developed definitions of response time. The development of this device was closely associated with the development of the analysis approach since they represented two parts of the same methodology.[8,9] In fact, the two types of development were totally integrated; when a problem was discovered by using the tool, a change in analysis approach often resulted, and changes in the tool frequently occurred as the methodology evolved.

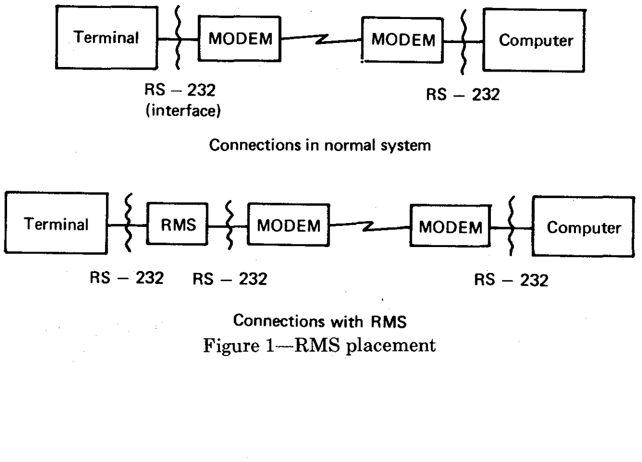
METHODOLOGY RESEARCH

Although our primary motivation was methodology development, we attempted to bias our work to be as applicable to normal situations as possible. We implemented a hardware device, the Rand Monitor/Stimulus-generator (RMS), that could be employed on any asynchronous system using the standard RS-232 interface. It interfaces between the terminal and communication lines as shown in Figure 1 and is a machine-independent device capable of stimulating the system with a prestored message and of measuring the response time for the message. It sends the same message repeatedly into the system by buffering the message and then sending it under clock control. The location of the device permits the terminal to be used for computer sign-on protocols and set-up procedures (by way of a bypass switch) as well as for input to the device buffers.

Maintaining simplicity of operation and providing readily analyzable data permits the analyst to concentrate on a performance problem rather than on the characteristics of the tool being used. The RMS, therefore, is not flexible enough to perform in all possible experimental designs or to collect all possible data on the operation of the system. However, simple tools such as the RMS can function quite satisfactorily if proper experimental design is used.[10,11]

Commands vs. scripts

Complexities inherent in other stimulation tools, due to their abilities to transmit elaborate scripts, were elimi-



Connections in normal system

Connections with RMS

Figure 1—RMS placement

* The test instrument was used during the development and later for the maintenance of the Rand Interface Block (RIB)[7] which interfaced all the consoles with the IBM 1800 process controller. It provides a facility to set the bit configurations representing up to three keyboard characters and three control words. Then, acting as a terminal, it sends the message to the RIB while the bit pattern is viewed through an oscilloscope. The message can be sent out once or continuously at varying speeds.

nated in our methodology by allowing transmission of only single-line messages. This not only eased design and use of the tool, but it led to simple experimental designs and results that are easily interpreted. Times for individual commands can then be added together to obtain script response time, and the projected variance can be computed.

The critical assumption is that the performance of the on-line system to a command is context-independent. That is, the response time of a command is independent of preceding commands. An example of a contrary case is the retrieval of a file; in some systems (e.g., Honeywell's TSS under GCOS) response time is shorter if the file has been referenced earlier during an on-line session. The analyst can ensure that this assumption is not restrictive by stating all such conditions and ensuring that they are met during tests.

Average response time vs. frequency of satisfactory responses

Identical average response times can be generated by entirely different distributions; user satisfaction will often differ among the distributions. In addition, an on-line user may not be able to distinguish between a response time of .1 second and one of .5 second; he may not care what the response time is as long as it is within 1.0 second. In such a case, a system with a constant response time of .9 second will be more satisfactory than an average response of .8 second resulting from 14 percent of the responses requiring 5 seconds and 86 percent requiring .1 second. A user who will leave the terminal if a response is in excess of 5 seconds does not distinguish between 10-second and 3-minute responses. A metric more meaningful than average response time for a specific command is the frequency of satisfactory responses. Examples of the use of this metric for a specific command might be: "faster than required" (less than 1.0 second), "satisfactory response" (1.0 seconds to 2.0 seconds), and "bad response" (over 2.0 seconds). The goal of an installation, with regard to its on-line system, should be to reduce the frequency of unsatisfactory responses. In addition to being more useful, this metric dramatically decreases the data reduction problems inherent in many performance analyses. Rather than collecting each response time for a specific command, the number of responses which occur within ranges specified by the analyst are collected.

Flexible definition of response time

There is no universal definition of response time. Figure 2 illustrates two possible definitions: carriage return to first character of the response* and carriage return to keyboard unlock. While arguments can favor either definition, the utility of each varies according to the actual command being measured. For some commands (particularly those which are data dependent, device dependent, or require operator intervention), the first character of the response is a reasonable indicator of response. A user may be anxious to see the first part of the response and willing to wait for the operation to be completed because of the characteristics (i.e., length, or complexity of the process). In contrast, a user inputting text is concerned with maintaining typing speed—an operation directly related to keyboard unlock. These two alternatives do not exhaust the possibilities; problems in specific systems may make other definitions more useful.

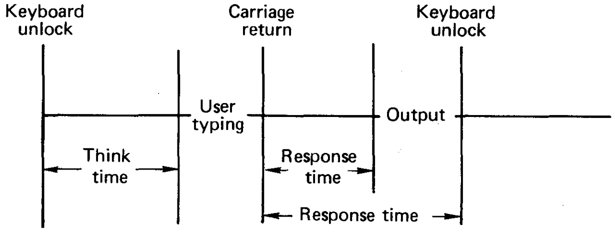


Figure 2—Response Times

Our early work on evaluating on-line systems indicated that a single, fixed definition of response would preclude investigating many on-line systems. A flexible technique for defining response was necessary to reflect the set of characters that indicate the beginning of a response. An arbitrary set of characters* is chosen to indicate the initiation of response timing.

An unexpected characteristic found in some on-line systems was that completion of user input may be indicated by characters other than a carriage return. For example, a single numeric character may constitute the total input from a user. In other cases a break character or some other single (or multiple) special character may indicate completion of user input and that the computer should take over control of the interaction. The initiation as well as the termination of the measured interval must be arbitrarily definable. For consistency, up to three characters indicate that the measurement clock be started or stopped.

THE RAND/MONITOR STIMULUS-GENERATOR

Modes of operation

The RMS operates under three modes: bypass, text entry, and transmit. When the monitor is in bypass mode, all inputs from the terminal pass directly to the communication line and conversely from the communication line to the terminal. This allows necessary protocols to be completed in a normal manner prior to monitoring.

Text entry mode allows the buffers to be filled with characters directly from the terminal. In addition to the

* This is usually stated as the first meaningful character—a somewhat ambiguous definition. J. Maranzano of Bell Telephone Laboratories has suggested the definition be directed toward the first character of the first output message.[12]

* A maximum of three characters was chosen—it has been adequate in all subsequent work.

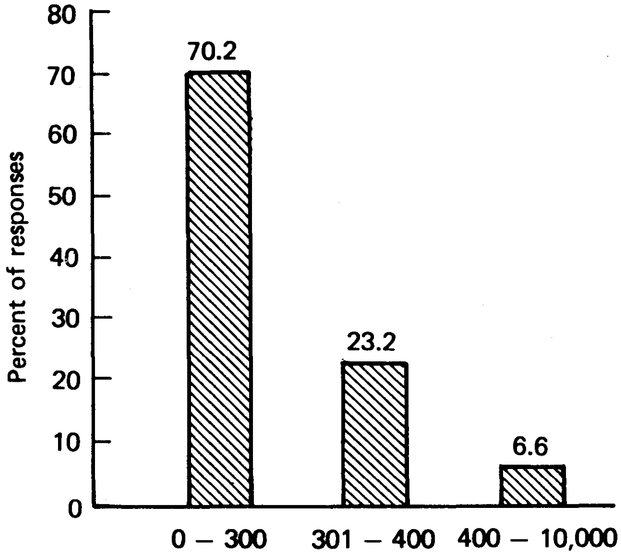


Figure 3—Response time (msec) of text addition

message to be sent, the characters designating "start timing" and those signaling the "stop timing" (possibly, first character back) are also input from the terminal.

Transmit mode is used for transmitting the stored message at settable line rates of 110 through 2400 baud to match the system's rate. System response is determined while in transmit mode. Transmission can be performed manually or automatically.

During transmission, characters are not necessarily sent out at the maximum speed allowed by the line rate. A restriction limiting transmission rate to the maximum might cause severe degradation of other users' ability to transmit over a common communication system. Similarly, automatically transmitting a new request immediately on receipt of a response could load the computer to the extent that other users would note the situation and act abnormally; this would eliminate the advantage of testing the on-line system under normal, natural loading. An arbitrary delay time can therefore be specified between each character, and a separate time can be specified to delay the automatic transmission of a new message after response to an old one (providing a simulated "think time").

Outputs

Numeric output of response time and status lights initially appeared adequate—response time being output only upon request (by depressing a button). It was quickly determined that the time of each response and the elapsed time for the current response (for responses exceeding a few seconds) were also of interest to the analyst.

The frequency counts and limits for each interval must be available during a test period for the analyst to determine when the session should be terminated. Human abilities to estimate response times and frequencies of the various ranges of response times were found to be very poor. Analysts have a tendency to believe that a test has gone astray due to very good (or bad) response times and terminate the test unless they can obtain assurance that all is well.

The rapidity of interactions in many tests taxes the abilities of analysts to understand what is happening. This problem was solved by installing a line of status lights indicating when the RMS was waiting to transmit, transmitting a request, waiting for a response, or receiving a response.

METHODOLOGY EVALUATION

Our proposed methodology for on-line system performance analysis asserts that an analysis can be performed by adding a marginal incremental load to the natural system load and measuring the response of the system to the incremental load. It further asserts that categorizing responses into a few ranges of responses enables the computation of meaningful metrics. Finally, it asserts that an inexpensive, hardware device can interface to a variety of different computer systems.

We evaluated these assertions by building the simple hardware device and successfully performing tests on three different computer systems (an IBM 360/65, an IBM 370/158, and a Honeywell 6050). The tests on the IBM machines evaluated the methodology as described and those on the Honeywell system evaluated the extensibility of the methodology.

Tests on IBM equipment

The tests on IBM equipment concerned evaluating the WYLBUR text editor system.[13] This evaluation was an extension of earlier investigations of WYLBUR response time.[6] One objective was to determine the stability of response time to commands for adding text to a working file.

Figure 3 shows the result of a data collection effort in

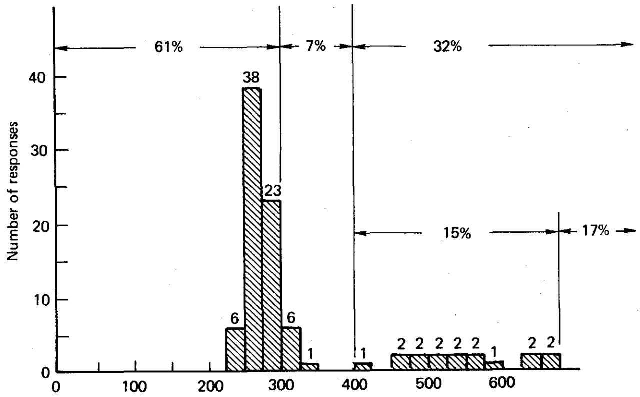
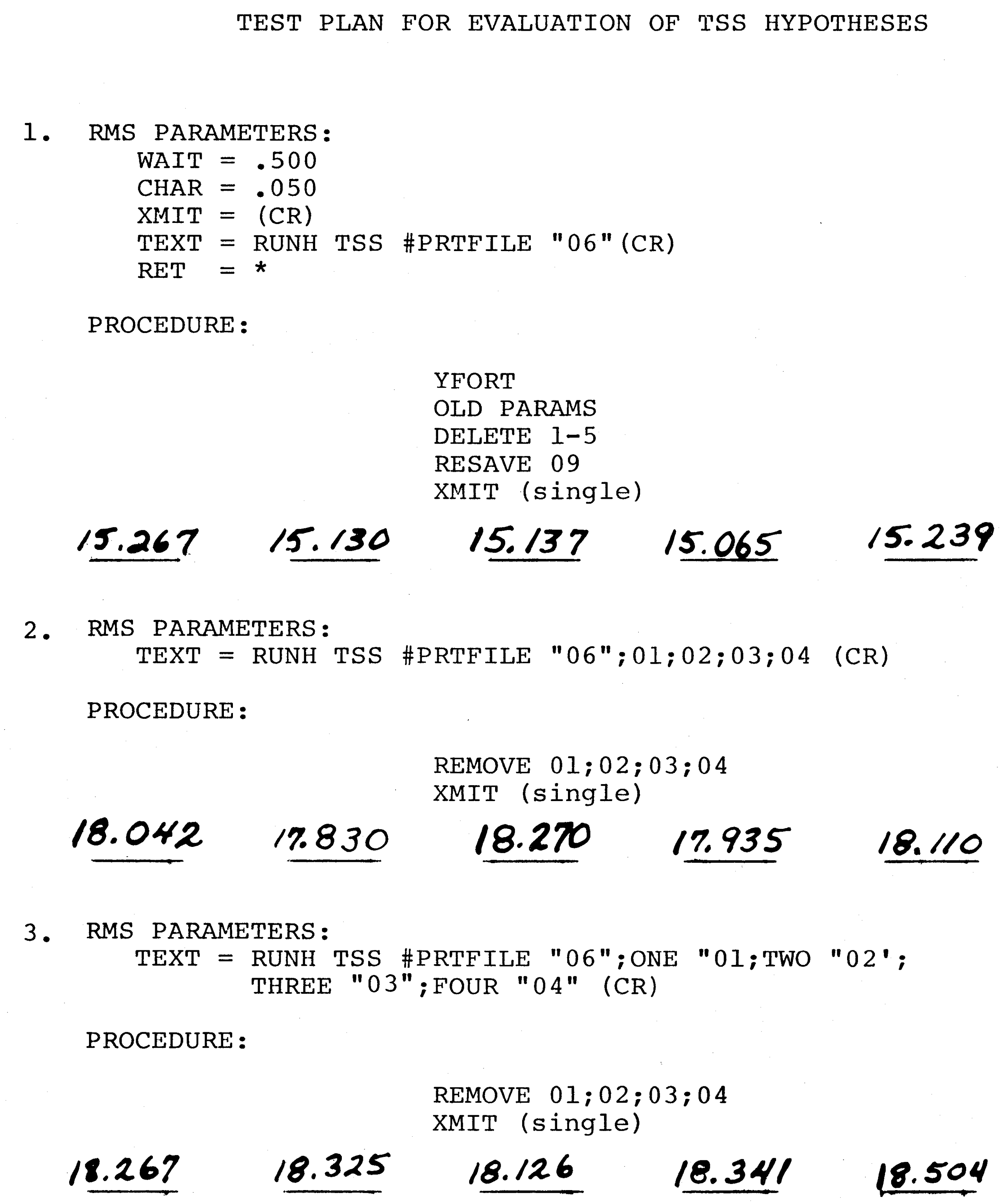


Figure 4—Response time (msec) of text addition



TEST PLAN FOR EVALUATION OF TSS HYPOTHESES

```
1.  RMS PARAMETERS:
       WAIT = .500
       CHAR = .050
       XMIT = (CR)
       TEXT = RUNH TSS #PRTFILE "06"(CR)
       RET  = *

    PROCEDURE:

                          YFORT
                          OLD PARAMS
                          DELETE 1-5
                          RESAVE 09
                          XMIT (single)
```

15.267 15.130 15.137 15.065 15.239

```
2.  RMS PARAMETERS:
       TEXT = RUNH TSS #PRTFILE "06";01;02;03;04 (CR)

    PROCEDURE:

                          REMOVE 01;02;03;04
                          XMIT (single)
```

18.042 17.830 18.270 17.935 18.110

```
3.  RMS PARAMETERS:
       TEXT = RUNH TSS #PRTFILE "06";ONE "01;TWO "02';
              THREE "03";FOUR "04" (CR)

    PROCEDURE:

                          REMOVE 01;02;03;04
                          XMIT (single)
```

18.267 18.325 18.126 18.341 18.504

Figure 5—Completed Form for Honeywell Tests

which the response time intervals were 0-300 milliseconds, 301-400 milliseconds, and over 400 milliseconds. Unlike the standard, prescribed procedure, each individual response time was recorded. Figure 4 shows the results in more detail; interactions with times in excess of 667 milliseconds were lumped together.

Introspection reveals that we had been more interested in collecting significant counts than in setting meaningful limits with respect to human interactions. Subjectively, the on-line system responded poorly during the data collection for Figure 4, but only 17 percent of the responses could be classed as annoying or unacceptable. Limits of 0.67 seconds and 3.0 seconds would have been far more appropriate. Only a few long responses cause users to perceive the system as slow; analysts should set limits to determine system acceptability.

Further observation indicated that user annoyance arose when several long responses occurred in a short interval. The relevant variable appears to be the number of long responses per interval of elapsed time. Computation of this metric would require dumping the counters at fixed intervals; an alternative might be to determine the number of long responses occurring within a prescribed interval after the first long response.

Tests on a Honeywell 6050

We also conducted a study which included the investigation of performance characteristics of a Honeywell 6050 computer. Part of the study involved measuring certain elements of on-line systems in a totally controlled environment. Measurements included initiation/termination time, processor time, and elapsed time of CPU-bound and I/O-bound jobs while varying file characteristics and runtime options. The series of tests employed the same synthetic benchmark job each time. This job produces data regarding elapsed time and processor time by interrogating the system internal clocks at appropriate intervals. These data were supplemented by those obtained from the RMS to determine the time of individual interactions.

The program was carefully designed so that the RMS could be used to measure the time from the carriage return to the first output character to approximate initiation time. We also used measurements of time terminating with keyboard unlocking (the appearance of an asterisk). To avoid having measured response time depend on the length of the output, speed of the terminal, and the buffering algorithm, we directed printed output onto a temporary disk file for subsequent printing. Figure 5 illustrates the procedure used for the tests and shows some of the data collected on the effects of file allocation on elapsed time. In this example, a CPU-bound job was run under three conditions: no files allocated, temporary files allocated, and permanent files allocated.

In summary, effort was expended to separate irrelevant effects from those of interest. In addition, a specially designed application job was required to coordinate internal operations with external effects. Without the RMS's flexible controlled input/response-timing capability, the tests could not have been performed successfully.

CONCLUSIONS

The methodology evaluation indicated that both the tool and the analysis approach performed well. The approach using a combination of natural loading and artificial loading proved to be effective, but occasional totally artificial loading also proved useful. Employing repeated, identical stimuli provided samples with known statistical properties, but we only partly solved the problem of metrics. In only a few cases could we find instances where even a much more powerful monitor/stimulus-generator would have made our evaluations easier; simple tools like the RMS appear to be quite cost/effective. Further work on the methodology would clearly be worthwhile, but immediate application of results is both feasible and justified.

As a result of concentrating on problem solution (rather than tool development) we produced an approach that has proved valuable in several empirical investigations. This simple approach depends on an inexpensive monitor/stimulator which we developed. The combined result is a cost/effective methodology to deal with the continuously important problem of tuning on-line systems.

REFERENCES

1. Bell, Thomas E., "Computer Measurement and Evaluation—Artistry, or Science?" *Performance Evaluation Review,* Vol. 1, No. 2, ACM, June 1972, pp. 4-10.
2. Abrams, Marshall D., George E. Lindamood, Thomas M. Pyke, Jr., "Measuring and Modeling Man-Computer Interaction," *First Annual SIGME Symposium on Measurement and Evaluation,* February 1973, pp. 136-142.
3. Lambert, David W., "TSO Workload Generation Using A Remote-Terminal Emulator," *Proceeding of SHARE XLII,* March 1974, pp. 68-86.
4. Baer, Peete, "Artificial Loading of On-Line Systems," *Proceedings of SHARE XXXVIII,* March 1972, pp. 224-230.
5. Tesdata Systems Corporation, 553 Wisconsin Avenue, Chevy Chase, Maryland 20015.
6. Lockett, J., "Computer Performance Analysis in Mixed On-Line/Batch Workloads," *AFIPS Conference Proceedings,* Vol. 43, National Computer Conference 1974, pp. 671-676.
7. Uncapher, K. W., *The Rand Videographics System—An Approach to a General User-Computer Graphic Communication System,* The Rand Corporation, R-753-ARPA, April 1971.
8. Lockett, J. and T. E. Bell, *The Rand Monitor Stimulus-generator: A Simple Approach to Response Time Analysis in On-Line Computer Systems,* The Rand Corporation, R-1649-PR, 1975.
9. Yoshimura, R., *The Rand Monitor Stimulus-generator: Hardware Implementation,* The Rand Corporation, R-1714-PR, (to be published).
10. Bell, T. E., B. W. Boehm and R. A. Watson, "Framework and Initial

Phases for Computer Performance Improvement," *AFIPS Conference Proceedings,* Vol. 41, Fall Joint Computer Conference 1972, pp. 1141-1154.

11. Shetler, A. C., "Controlled Testing or Computer Performance Evaluation," *AFIPS Conference Proceedings,* Vol. 43, National Computer Conference, 1974, pp. 693–699.
12. Maranzano, J. F., "Proposal for a Definition of Response Time," *SHARE Computer Measurement and Evaluation,* Vol. II, 1974, pp. 484-496.
13. Fajman, R. and J. Borgelt, "WYLBUR: An Interactive Text Editing and Remote Job Entry System," *Comm. of ACM,* Vol. 16, No. 5, May 1973, pp. 314-322.

A heuristic approach to computer systems performance improvement, I—A fast performance prediction tool*

by STEPHEN R. KIMBLETON**
USC/Information Sciences Institute
Marina del Rey, California

INTRODUCTION

The design, sizing and tuning of computer systems is a persistent, expensive, time-consuming, and difficult problem. An appropriate, integrated support methodology to support the needs of the computer center manager vis-a-vis these issues is lacking. Although the performance literature related to this topic is voluminous, much of it is directed to the vendor rather than the computer center manager.

Development of a methodology to support the computer center manager requires explicit consideration of the large variety of constraints which must be observed and should support: (1) determination of the reasonableness of existing performance, i.e., 'glitch' detection, (2) evaluation of the effects of system modifications and alterations, and (3) projection of the performance of new systems or systems with redefined functional requirements. The complexity of systems, constraints, and the resulting decision-making process precludes algorithmic approaches to 'good' system design. The well-structured nature of first order technological decisions, which can be divided into four categories: (D1) hardware configuration, (D2) software capabilities, (D3) job schedules, and (D4) file-device assignments, suggests the feasibility of implementing heuristic (informed trial-and-error) approaches. The ability of such approaches to effectively solve complex problems is apparent from the quality of current chess playing programs.

Heuristic approaches require three capabilities: (1) performance prediction to determine the exact performance of a given system processing a specified workload in accord with a given schedule, (2) system comparison to determine the more desirable of two system alternatives in the context of a given schedule and workload, and (3) stopping time determination for terminating the iteration cycle implicitly defined by (1) and (2).

This paper describes one approach to obtaining a very fast performance prediction methodology suitable for supporting heuristic systems design. Possible comparison criteria for scheduling batch computer systems are described in Reference 1. Identification of stopping time mechanisms can be safely deferred pending completion of a thorough exploration of the feasibility of this approach.

PERFORMANCE PREDICTION REQUIREMENTS

Development of a performance prediction mechanism requires identification of: (1) objectives and information requirements, (2) accuracy requirements, (3) level of detail, and (4) modeling technology. In this section, goals for each of these categories will be detailed. The remainder of the paper may be viewed as a demonstration of the feasibility of their achievement.

Objectives and information requirements

Technological control decisions are reflected in the four decision categories (D1)-(D4) cited above. It follows that evaluation of alternatives requires knowledge of individual device utilizations and delays. Further, it is desirable that this information be broken out on a per job basis, a per shift basis, and as will be seen later, a per time segment (period of time during which the composition of the mix remains constant) basis.

Accuracy requirements

The difficulty of designing and implementing a simulator is a function of the accuracy level required. The user of a simulator naturally seeks a very high accuracy level, say 1 percent (that is |obs-pred|/obs<.01 where obs denotes the observed value of a statistic and pred denotes the value of the statistic predicted through usage of the simulator). Vendors of current, commercially available simulators use persuasive salesmanship to convince computer

* Preparation of this paper was supported by the Advanced Research Projects Agency under Contract No. DAHC15 72 C 0308, ARPA Order No. 2223, Program Code No. 3D30 and 3P10, and the Office of Naval Research, Information Systems Program under Contract N00014-67-A-0181-0036 (NR 049-311).

** The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency, the Office of Naval Research, or the U.S. Government.

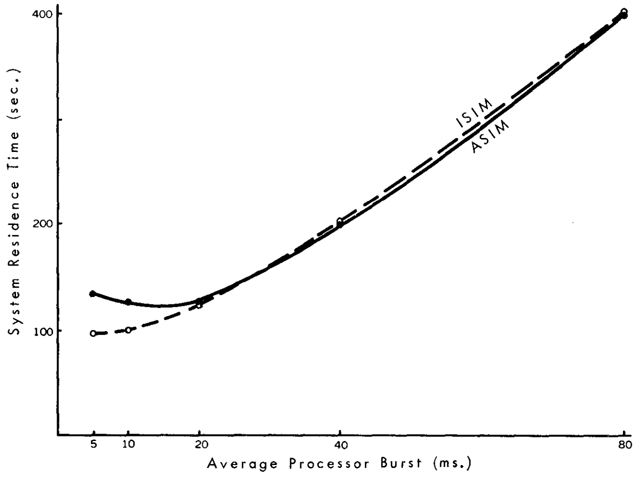


Figure 1—ASIM-ISIM system residence time comparison

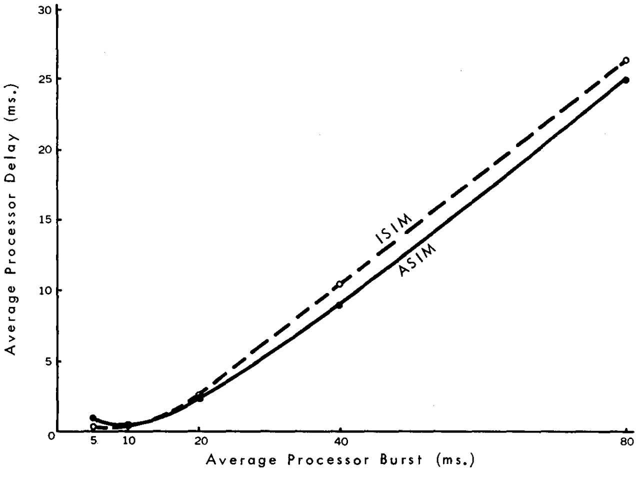


Figure 3—ASIM-ISIM processor delay comparison

center management that accuracy levels of 1 percent are readily obtainable—although the tuning required to achieve this level of accuracy mitigates against the utility of the simulator in investigating new configurations or in evaluating the reasonableness of existing performance (two primary uses for a simulator). Acceptance of a lower level of accuracy requires education of the user through identification of accrued advantages which typically include: reduction/elimination of the need for calibration, increased speed of execution, and expedition of the verification/validation process. Based upon discussions with managers, it is the author's conjecture that most users can easily tolerate an accuracy level of 20 percent. This paper hypothesizes, and partially verifies, that such an accuracy level can be achieved through usage of fast, analytically driven techniques.

Level of detail

Performance prediction techniques can be classified in terms of the basic level of detail incorporated: hardware (register), operating system, resource allocation, time segment, and job. Decreasing the level of detail decreases the potentially achievable level of accuracy as well as the cost of implementation and cost of verification/validation of the simulation. Our objective is to achieve an accuracy level of 20 percent, and our assertion is that this level of accuracy can be achieved through prediction of time segment performance and aggregation of the resulting statistics to achieve job and shift performance. This assertion is unprovable. However, supporting evidence is provided in the comparisons detailed in Figures 1-5, included at the end of the paper. Although the 20 percent discrepancy level is not achieved in all cases, non-achievement seems clearly due to usage of an approximation technique providing only lower bound estimators for processor utilization. Subsequent efforts directed to achievement of the desired tolerance level through development of improved processor utilization estimators seem very likely to succeed. Moreover, it will be noted that the level of discrepancy decreases as the processor utilization increases.

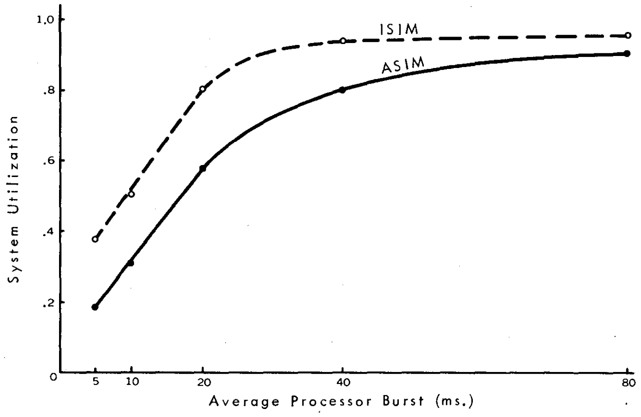


Figure 2—ASIM-ISIM cost weighted system utilization comparison

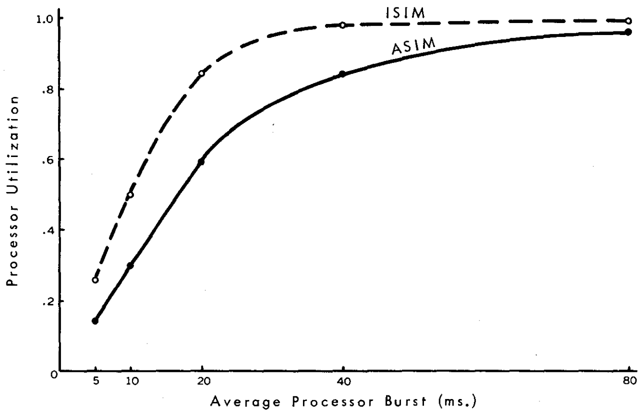


Figure 4—ASIM-ISIM processor utilization comparison

Modeling technology

Prediction of system performance can be approached through either analytic or simulation techniques. Because of the limited information available from an individual analytic tool, analytic approaches have been relegated to the role of design tools used in gaining insight into subsystem tradeoffs and policy. Additionally, analytic models are occasionally used to estimate gross system performance characteristics through extensive aggregation of system components which is required for computational feasibility.[2,3] The advantage of analytic approaches is their speed of implementation and execution coupled with relatively low cost for developed models; the disadvantages include extensive simplifying assumptions, the feasibility of incorporating only a meager level of detail, the difficulty of understanding them without extensive training in modeling techniques, and the prevailing lack of reasonable user interfaces.

Simulation approaches, in contrast, permit incorporation of almost any desired level of detail. Since there is a prevailing tendency to incorporate superfluous factors, and a great discrepancy between the rate at which events occur in a computer system (micro seconds—milli seconds) and the run time of a computer system (hours—days), most simulation models execute relatively slowly, e.g., 1-10 times faster than real-time.

Analytic and simulation approaches can be regarded as two endpoints of a spectrum of possible approaches to computer system performance prediction. Intermediate points within this spectrum would blend simulation and analytic approaches in a desire to balance execution speed, output information, required input data, and accuracy. The objective of this paper is to demonstrate that through proper development of an analytically driven approach, information essentially equivalent to that provided through current, commercially available simulators, can be achieved very quickly. Thus, the basic performance prediction requirement for implementation of a heuristic approach to computer system design, sizing, and tuning will have been achieved.

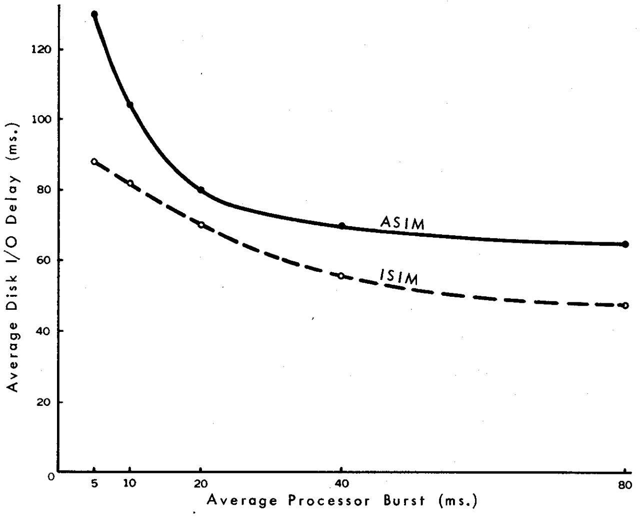


Figure 5—ASIM-ISIM disk I/O delay comparison

ANALYTIC PREDICTION OF SYSTEM PERFORMANCE

Development of an analytic technique for predicting computer system performance requires identification of: (1) the precise analytic techniques to be used, (2) the input data required, (3) the output information sought, and (4) the techniques to be used to derive (3) from (2). Item (4) proves not to be straightforward since a variety of analytic techniques must be interrelated to obtain a capability which can provide output information competitive with that provided by a resource allocation level simulator. Each of these issues will now be discussed in turn.

Selection of basic analytic approach

Two major approaches to modeling processor-I/O interaction within a computer system can be differentiated through their representation of I/O delay. The queuing network approach represents I/O devices by their service time and obtains global system results (device utilizations and delays) in terms of these service times and the transition matrix guiding migration among I/O devices and the processor (also represented in terms of its service time). Alternative analytic approaches represent I/O devices in terms of total I/O delay and are provided by the machine repair model and an approximation technique termed the system process model.[4]

Wide usage of the queuing network approach has been made in computer network analysis,[5] communications system design,[6] and computer systems analysis.[7,8,2] The results obtained via this approach have proved encouraging and extensive research is continuing on a variety of related topics. Usage of this approach as the kernel of an analytically driven performance prediction mechanism proves difficult for two reasons: (1) representation of data path delays, and (2) computational complexity. The first reason reflects the fact that incorporation of data paths requires a mathematical capability to handle queuing networks with correlated service times—a capability which currently appears to be in advance of the state of the art. The second reason reflects the fact that the complexity of the queuing network approach increases with the square of the number of nodes incorporated. Thus, published studies have found lumping of nodes necessary to reduce the dimensionality from the 50 or more resources comprising the typical large scale computer system to 5-10.[2]

In I/O delay approaches, the input information is mean processor burst and mean I/O delay; the output information is processor utilization and delay. The standard machine repair model assumes exponentially distributed variables and an iterative technique has been developed[9] for calculating the relevant statistics under the assumption

TABLE 1—Job/System Description (JSD) Components

Exogenous JSD Variables

1. Name
2. Time of Arrival
3. Static Priority
4. Job Due Date
5. Job Setup Time
6. Job Precedence Relations
7. Number of Job Steps

Endogenous JSD Variables

1. Processor Interaction
    1. Job Processor Requirements
    2. Processor Burst Description
    3. CPU/I/O Overlap
2. Memory Requirements
3. File List
    1. Number of Files Accessed by Job
    2. File Names
    3. File Access Count
    4. Data Transfer Size
4. File Characteristics (For Each Listed File)
    1. File Device Location
    2. Latency
    3. Seek Time Distribution (for disks)
    4. File Size
    5. File Organization

of non-exponentially distributed I/O delays. This technique has been successfully applied to computer system modeling in which the workload is characterized by an 'average' job(s).[10]

The system process model[4] uses a similar job characterization. Exponential assumptions are shown to constitute a natural midpoint for a wide range of distributional assumptions. The estimator of processor delay is independent of the approximation used to obtain processor utilization; thus, alternative techniques for estimating processor utilization, including the machine repair approach, can be used. This computational technique has been used because of the straightforward nature of the calculations which constitute the kernel computation used in developing an analytically driven performance prediction technique termed ASIM.

I/O delay approaches naturally enable a hierarchical approach to calculation of system statistics in which I/O delay is first determined followed by evaluation of other system performance characteristics. Since, as shown in the following section, the kernel calculation for I/O delay and processor utilization is relatively straightforward, the complexity of the total computation is approximately linear in the number of devices comprising the system, and is thus competitive with simulation approaches for both large and small systems.

DESCRIPTION OF AN ANALYTICALLY DRIVEN PERFORMANCE PREDICTION TOOL (ASIM)

The major objective in the development of ASIM was fast prediction of the performance of a given system executing a specified collection of jobs in accord with a defined schedule. In this section we describe: (1) simplifying assumptions used in the initial ASIM implementation, (2) input data characterization, (3) output information developed, (4) conceptual steps involved in obtaining system performance statistics, and (5) accuracy/timing considerations.

ASIM implementation assumptions

The following eight simplifying assumptions were chosen to represent a compromise between code simplification and demonstrating feasibility appropriate to actual utilization in a computing environment: (1) a single processor system, (2) one data path per device, (3) zero setup times for jobs (both initiation/termination and disk/tape mount/dismount times), (4) all jobs have equal priority, (5) one active data set per device per job, (6) no unit record equipment, (7) jobs use a fixed amount of user memory, and (8) a maximum of thirty drums, disks, and tapes; all drums are shared, a user specified number of disks may be shared, and all tapes are non-shared.

Removal of the single processor assumption is possible at the relatively minor cost of requiring another (internal) list for each additional processor and incorporation of an estimator for processor contention impact on system performance. Explicit incorporation of data paths requires development of a suitable approach to representation of data path induced delay; existing models in the literature are oriented to static prediction and inappropriate to the objectives of this model. Removal of the remaining assumptions is a programming exercise (note that average working set size can be used for virtual memory systems in place of partition size for those systems based on a working set philosophy).

ASIM input data

The input data to ASIM consist of three files: (1) Job/System Descriptions, (2) System Characteristics, and (3) Switch Settings. Switch settings constitute a superset of the standard GASP switch settings[11] which initialize a run and will not be discussed further.

Table I indicates the components of a Job/System Description (JSD). As is apparent, a JSD is related to a synthetic module characterization of a job[12-14] and consists of two components. The exogenous component describes the environment defined for the job and its relation to other jobs, while the endogenous component describes the resource requirements and utilization rates for an individual job. Although automatic generation of JSD's is not currently feasible, careful examination of the more sophisticated accounting systems demonstrates the feasibility of capturing these data through minor modifications. Indeed, SMF data are currently being utilized to generate input data for some currently available simulators.[15]

It should be noted that the JSD was so named since it constitutes a viable description of the job only in the context of a given target system. In particular, no assertion is intended that the same source code executed on systems produced by two different vendors will produce the same JSD's.

Jobs are assumed to be initiated in the order indicated by the list of JSD's subject to three constraints: precedence satisfaction, arrival time satisfaction (a job cannot be initiated prior to arrival), and resource availability. If one of these constraints is unsatisfied, initiation is deferred pending its satisfaction. Further, no attempt is made to look ahead in the JSD sequence to determine if another job could be initiated. It should be noted that this requirement is necessary in order to permit study of computer system performance as a function of the schedule.[1] It does not reflect current job scheduling in which both the operating system and the external scheduling mechanism jointly share responsibility. Although the current approach to system scheduling has proven useful (perhaps unavoidable), in practice it greatly impedes determination of the performance of a given computer system executing jobs in accord with a prescribed schedule. Accordingly, the approach which we have described has been implemented. Modification of this approach to more accurately reflect the characteristics of scheduling as implemented on an individual computer system is essentially a programming problem.

Table II indicates the collection of information comprising the system characteristics. Their determination is straightforward.

ASIM output information

Output information generated by ASIM is aggregated into three categories: (1) job performance statistics, (2) shift performance statistics, and (3) time segment performance statistics. The information in each category consists of device utilizations and delays as shown in Table III together with some header information which is not given. For a job this header information includes job identification and for a time segment the collection of jobs in concurrent execution is displayed.

TABLE II—System Characteristics

Number of Drums	0*
Number of Disk Drives	3
Number of Tape Drives	0
Number of Non-Shared Disk Drives	0
Amount of Core Memory	100
Rotation Time of Drum	35.0000
Number of Pages Per Drum Track	6
Rotation Time of Disk	16.6667
Number of Pages Per Disk Track	5
Total Number of Cylinders	411
Minimum Seek Time	10.0000
Maximum Seek Time	55.0000
Average Seek Time	30.0000
Tape Interrecord Gap	0
Cost Per Unit Time of CPU	10.000000
Cost of Core Per Unit Time	1.000000
Cost of Drum Per Unit Time	0.015000
Cost of Disk Per Unit Time	0.006800
Cost of Tape Per Unit Time	0.005000

* Numbers are representative values used in the example.

TABLE III—ASIM Output Information

Processor Utilization
Processor Delay

Memory Utilization

Average Degree of Multiprogramming
Average I/O Delay
Average System Utilization

Individual Drum Utilization
Individual Drum Delay
Individual Drum Service Time

Individual Disk Utilization
Individual Disk Delay
Individual Disk Service Time

Individual Tape Utilization
Individual Tape Delay
Individual Tape Service Time

ASIM performance calculations

Let an event denote the time at which either a job initiation or job termination occurs and let a time segment denote the time between two successive events. For a shift in which N jobs are processed, there will be at most $2N$ events and $2N$-1 time segments; the upper bound is reached for non-concurrent initiations. Since the composition of the mix is constant over a time segment, analytic prediction of time segment performance is in order. Standard aggregation techniques used for statistics accumulation in simulation can then be used to develop job and shift performance. Additionally, time segment performance is captured since poor performance for a job implies poor performance for at least one time segment. Further, development of heuristic scheduling procedures requires knowledge of time segment performance characteristics[1] to evaluate the desirability of alternative job interchanges in heuristically seeking a 'good' schedule.

Label the collection of jobs to be processed during a representative time segment $J1, \cdots, Jn$. Generation of processor delay and utilization requires generation of an average active and blocked interval for this time segment appropriate to this collection of jobs.

Determination of average I/O delay requires care. In its calculation, an average processor burst which is simply the arithmetic average of the processor bursts for jobs $J1, \cdots,$ Jn, is used. It is evident from the Job/System Descriptions that knowledge of the JSD's for $J1, \cdots, Jn$ permits straightforward calculation of the probability of a reference to an individual device during the time segment (assuming accesses are equally probable over a time segment). This probability, together with the average processor burst described above, permits a mathematical determination of upper and lower bounds for the interarrival times to a given device. (Determination of the mean interarrival time proves difficult since it is affected by device characteristics and queues; thus, its determination requires an iterative approach and was deferred in the interest of computational speed.) Given these bounds, bounds for device delay can then be obtained through queuing calculations.

Usage of available device models within the literature is a natural objective which is precluded by their assumption of an infinite calling population. In practice, for batch systems, a degree of multiprogramming in the range of 3-6 has been typical.[16] Further, the number of 'batch equivalents' in interactive systems has been observed to fall in this range since an average batch job is approximately the same system load as 10 average interactive jobs.[2] Clearly, such estimators are at best crude. Moreover, data on batch equivalents for non-university environments is not readily available although similar computational approaches can be used.

Because of the unsuitability of existing I/O device models, a technique described in Reference 17 for obtaining the steady state queuing time distribution for the M/G/1 finite capacity queue proved appropriate since it effectively proceeds by computation of results for an associated cyclic two server queue in which one server represents the processor and the other can be thought of as a generalized I/O device. Although data paths are not explicitly incorporated in the present version of ASIM, their incorporation is feasible in the approach described and an approximately linear complexity in terms of the number of devices is retained.

Having obtained the delay for an individual device, average delay for all devices can then be obtained. This then permits determination of processor utilization U and the processor delay D via the following theorem:[18]

Theorem:

Let A and B denote the average length of the active and blocked intervals of the process. For $K \geq 2$ concurrently executing statistically identical processes:

(i) $U > 1 - \rho K - 1$
(ii) $D = (K - J) A U(K)$

where $\rho = B/(A+B)$, and $J = B/A$ denotes the expected number of active intervals which can be initiated during a blocked interval.

Application of this theorem and the I/O results described earlier yields all device delays and utilizations. Determination of core utilization is a simple calculation.

ASIM timing and implementation considerations

For portability reasons, ASIM has been implemented in a Fortran superset GASP.[11] Although GASP is a simulation language, it was used for its output formatting and report generation capabilities rather than its list manipulation capabilities. Retrospect suggests that direct coding of desired output capabilities and elimination of the GASP routines is feasible and would ease portability.

Description of ASIM speed is misleading since it is a function primarily of the number of jobs executed during a shift rather than their duration. Thus, execution time is the sum of the time segment execution times which are effectively linear in the number of distinct devices referenced during the time segment. Consequently, overall execution time is nearly linear as a function of the number of jobs and distinct I/O devices. Typically, a time segment performance calculation requires approximately one second of processor time on a medium-sized computer such as TENEX. Thus, determination of performance statistics for a shift with 50 jobs requires less than two minutes of processor time. Although most shifts process a significantly greater number of jobs, many of these jobs require only very limited amounts of resources and are better represented in the aggregate rather than individually, in view of assumption 3.

Although the description of ASIM calculations is relatively straightforward, this reflects hindsight and is not represented in the current version which consists of approximately 1800 source statements. Knuth's remark to the effect that the best way to implement a program is to code it once, throw it away and code it again seems very appropriate. A new version is now being coded and the objective is to reduce the number of lines of source code to approximately 500. The next section provides some comparison information for ASIM evaluation.

AN EXAMPLE AND SOME DIRECTIONS FOR FUTURE RESEARCH

Evaluation of ASIM is easier than evaluation of a simulator since the objective is to demonstrate that information approximately as detailed as that obtained via simulators can be obtained while a significant increase in speed is also obtained. For comparison purposes, a resource allocation simulator (ISIM)[19] was used. Figures 1-5 provide comparison information. These comparison statistics were obtained for five identical concurrently executing jobs accessing three disks in a uniform manner and differing only in the length of the average processor burst which was 5, 10, 20, 40 and 80 ms. Each job requires 1000 processor bursts, arrived at time 0, had no precedence or due date constraints, and was sufficiently undemanding in core requirements to permit simultaneous initiation.

Figure 1 indicates that the correlation between ASIM and ISIM predictions (for a system whose characteristics are as indicated in Table II) was very good for gross statistics such as the system residence time (elapsed time from initiation to termination of job). Further, Figure 2 shows a reasonable correlation between overall system utilization for ASIM and ISIM as represented in the system reward function (a dollar weighted device utilization statistic computed over all four device categories). Figure 3 demonstrates that the average processor delay also compares well. Figure 4 shows that the comparison between processor utilizations is not as satisfactory as is also true for the comparison between disk delays as shown in Figure 5. This discrepancy reflects the fact that for low access rates to I/O devices, the gap between the computed upper and lower delay bounds for individual devices is fairly wide. Indeed, the envelope between these two bounds contained the ISIM prediction in all cases. Thus, a refinement of the I/O delay prediction technique is required and is perhaps the most important improvement needed. A major requirement for achieving this accuracy improvement is developing an improved estimator for processor utilization, as is apparent from Figure 4.

Collectively, this information indicates that the desired level of accuracy has been achieved for gross overall statistics and that some refinement is needed for the device statistics. Thus, we argue that the basic feasibility of obtaining a 20 percent accuracy level has been shown and effort directed toward consolidation and improvement is merited. It is natural to consider the quality of the comparisons for non-statistically identical jobs. Surprisingly, the comparisons were slightly better and, in general, the accuracy of the predictions seems to improve with increasing system complexity.

Results obtained to date support the hypothesis that analytically driven performance prediction techniques providing significant speed advantages over those obtainable via a simulator can be constructed. Further, the discrepancies between ASIM and ISIM appear due to inadequacies in existing analytic techniques rather than to fundamental deficiencies in the approach. Extensive exploration of the limits of this approach is clearly desirable.

Discussion of ASIM speed is misleading since it is effectively independent of the length of the job. For production batch environments in which the average job requires five minutes or more to execute, a claim of two orders of magnitude faster than real-time is reasonable. For a university environment in which the average job requires two seconds of processor time this claim is clearly inappropriate; thus the restriction to production batch environments.

Usage of ASIM is not restricted to batch environments provided that one is willing to represent the interactive load through an 'average' approach. Because the standard deviation of resource requests for interactive jobs is significantly larger than the mean,[2] development of upper and lower bounds seems more appropriate.

The capabilities provided by ASIM permit essentially distinct approaches to two important issues: (1) computer system scheduling via heuristics as discussed in Reference 1, and (2) centralized design and control of large networked systems as discussed in Reference 20. The feasibility of achieving practically usable results in either of these two areas is increased by the possibility of automatic gathering of ASIM input data through minor modifications to the accounting system on most large computer systems. Indeed, the information required can currently be gathered through post-processing of information provided by accounting tapes.[15]

ACKNOWLEDGMENTS

The author would like to express his appreciation to Dr. Thomas E. Bell who critically reviewed an earlier draft of this paper, and to Mr. James Cochran who transferred ASIM to the Institute TENEX computer system and executed all experimental runs.

REFERENCES

1. Kimbleton, S. R., "Batch Computer Scheduling: A Heuristically Motivated Approach," *Proceedings of the Second Annual SIGMETRICS Symposium on Measurement and Evaluation,* Montreal, Canada, September 30—October 2, 1974.
2. Moore, Charles G., *Network Models for Large-Scale Time-Sharing Systems,* Technical Report No. 71-1, Department of Industrial Engineering, The University of Michigan, Ann Arbor, Michigan, 1971.
3. Hanssmann, F., W. Kistler and H. Schulz, "Modeling for Computer Center Planning," *IBM Systems Journal,* Vol. 10, No. 4, 1971, pp. 305-324.
4. Kimbleton, S. R., *An Approximate Analytic Technique for Hierarchical Computer System Modeling,* ISI/RR-75-30, to appear April, 1975.
5. Kleinrock, L., *Communication Nets,* McGraw-Hill Book Company, Inc., New York, 1972.
6. Kleinrock, L., "Analytic and Simulation Methods in Computer Network Design," *AFIPS Conference Proceedings,* 1970 Spring Joint Computer Conference, pp. 569-579.
7. Baskett, F., "The Dependence of Computer System Queues Upon Processing Time Distribution and Central Processor Scheduling," *Proceedings of the Third Symposium on Operating Systems Principles,* October 1971, pp. 109-113.
8. Buzen, J., "Analysis of System Bottlenecks Using a Queuing Network Model," *Proceedings of ACM (SIGOPS) Workshop on System Performance Evaluation,* Harvard University, April 5-7, 1971, pp. 82-103.
9. Gaver, D. P., "Probability Models for Multiprogramming Computer Systems," *Journal of the ACM,* Vol. 14, No. 3, July 1967, pp. 423-438.
10. Sekino, A., *Performance Evaluation of a Multiprogrammed Time-Shared Computer System,* Ph.D. Thesis, MIT-Lincoln Laboratory, August 1972.
11. Pritsker, A. A. B. and P. J. Kiviat, *Simulation with GASP II,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1969.
12. Buchholz, W., "A Synthetic Job for Measuring System Performance," *IBM Systems Journal,* Vol. 8, No. 4, 1969, pp. 309-318.
13. Hamilton, P. A. and B. W. Kernigan, "Synthetically Generated Performance Test Loads for Operating Systems," *Proceedings of the ACM-SIGME Symposium on Measurement and Evaluation,* February 26-28, 1973, pp. 121-126.
14. Sreenivasan, K. and A. J. Kielnman, "On the Construction of a Representative Synthetic Workload," *Communications of the ACM,* Vol. 27, No. 3, 1974, pp. 127-132.
15. Edgecomb, G., *SCERT Model Building System. User's Guide,* Chemical Abstracts Service, Columbus, Ohio, September 1973, PB 234 934.
16. Rodriguez-Rosell, J. and J. Dupuy, "Instrumentation for the Evaluation of a Time-Sharing, Page Demand System," *AFIPS Conference Proceedings,* 1972 Spring Joint Computer Conference, pp. 759-766.
17. Lavenberg, S. S., *The Steady-State Queuing Time Distribution for the M/G/I Finite Capacity Queue,* IBM Research Report RJ 1150, San Jose, California, 1973.

18. Kimbleton, S. R., "Performance Evaluation—A Structured Approach," *AFIPS Conference Proceedings,* 1972 Spring Joint Computer Conference, pp. 411-416.
19. Kimbleton, S. R., *Interrupt-Free Computer System Simulator (ISIM) Description,* USC/Information Sciences Institute Internal Document, 1974.
20. Kimbleton, S. R., "Modeling Considerations in Computer Communication Resource Control," *Proceedings of the Eighth Hawaii International Conference on System Sciences,* January 7-9, 1975, Honolulu, Hawaii, pp. 95-97.
21. Barlow, R. E. and F. Proschan, *Mathematical Theory of Reliability,* John Wiley & Sons, Inc., New York, 1967.
22. Butler, M. K. and A. Hirsch, *Use of the Argonne Benchmark Collection to Measure Computer System Performance,* Argonne National Laboratory, Applied Mathematics Division, ANL-8127, September 1974.
23. Couger, J. D., "Evolution of Business System Analysis Techniques," *Computing Surveys,* Vol. 5, No. 3, September 1973, pp. 167-198.
24. Kobayashi, H., "Application of the Diffusion Approximation to Queuing Networks I: Equilibrium Queue Distributions," *Journal of the ACM,* Vol. 21, No. 2, 1974, pp. 316-328.
25. Lucas, H. C., "Performance Evaluation and Monitoring," *Computing Surveys,* Vol. 3, No. 3, September, 1971, pp. 79-92.
26. Muntz, R. R. and J. W. Wong, "Efficient Computational Procedures for Closed Queuing Network Models," *Proceedings of the Seventh Hawaii International Conference on System Sciences,* University of Hawaii, Honolulu, 1974.
27. Organick, E. I., *The Multics System: An Examination of its Structure,* MIT Press, Cambridge, Mass., 1972.
28. Boole and Babbage, Problem Program Evaluator, 1970.
29. Teichroew, D. and H. Sayani, "Automation of System Building," *Datamation,* August 15, 1971, pp. 25-30.

Computer design verification via software simulation

by R. E. KARNES and W. A. CARTER
IBM Federal Systems Division
Huntsville, Alabama

INTRODUCTION

This paper describes the simulation approach* utilized in the design of a Large Scale Integration (LSI) computer. The objectives to be fulfilled via software simulation were verification of the computer design and the associated micro-code. Since no Vendor Technology Logic (VTL) breadboard was planned, the fulfillment of the objectives was critical to the success of the total design effort. The simulation approach supported not only the initial computer design and introduction of micro-code to the design but also the verification of the design as implemented in detailed logic.

A computer developed via the simulation approach described in this paper is the Hybrid Technology Computer (HTC), a member of NASA's Space Ultra-Reliable Modular Computer (SUMC) family. The HTC is a micro-program controlled, fixed point, general purpose computer whose design employs LSI circuit technology and packaging tailored for advanced space applications. Logic technology for the HTC utilizes low power, bipolar circuitry contained in LSI chips with a capacity for 134 gates per chip. Figure 1 is a functional block diagram of a HTC.

SIMULATION APPROACH

The simulation approach utilizes both a high level mode of simulation and a low level mode of simulation. The high level mode of simulation supports programs written in a high level language similar to PL/I. These programs, called models, describe hardware behavior at a black box or functional level. Two examples of high level simulation models are shown in Figures 2 and 3. Figure 2 is a model of a hardware register. When the clock line changes from '1' state to '0' state and the register select line is a '1', the register delays a time amount estimated by the designer and then gates its output. Figure 3 is a model of Instruction Read Only Memory (IROM). When the address lines change, an array that simulates the read only memory contents is accessed and the data is output 100 ns later.

These two examples are illustrated here because they are relatively simple and easily convey typical model construction and function. It should be realized, however, that there are some models containing complex logic and requiring as many as 400 high level statements. Within the scope of the simulation, 35 high level models were used that described the HTC. Table I identifies the models required to simulate the HTC, and lists the number of inputs and outputs and high level statements associated with each model.

Low level simulation is simulation of the designer's logic at a logic block or gate level. The designer's output is LSI chip logic diagrams with up to 134 gates of logic. The logic diagrams for a chip form the low level simulation building block. Each low level chip design corresponds functionally to a high level model. The software also supports a mix of high level and low level simulation by providing facilities to interconnect the input/output lines of high level models with the input/output lines of the low level chip logic.

Using the high level simulation capabilities, the HTC design was described entirely with the high level models. This high level model of the HTC allowed introduction of the micro-code to the hardware design early in the development cycle and provided a method for finding design problems and incompatibilities much earlier than a hardware breadboard would have allowed since chip logic design was not complete at this time. Utilizing the high level model, HTC diagnostic programs were simulated and the full computer instruction set verified.

As the chips' low level (detailed logic) design progressed and became available, they were integrated into the simulation process. This was done in a step by step procedure. When a chip's detailed logic was complete, the corresponding high level model was taken out and the chip's detailed logic replaced its function in the simulator. This replacement procedure allowed a gradual yet continuing detailed verification of the computer design as the detailed logic became available. This process continued until the actual logic of the machine was implemented and verified and all the chip designs released to manufacturing.

The final detailed simulation of the HTC consisted of a total of 55 low level chips and 10 high level models. The high level models that remained in the simulation were primarily vendor devices, such as high speed memories, that required no detailed verification.

* Work sponsored by the National Aeronautics and Space Administration, Marshall Space Flight Center, under Contract NAS8-14000.

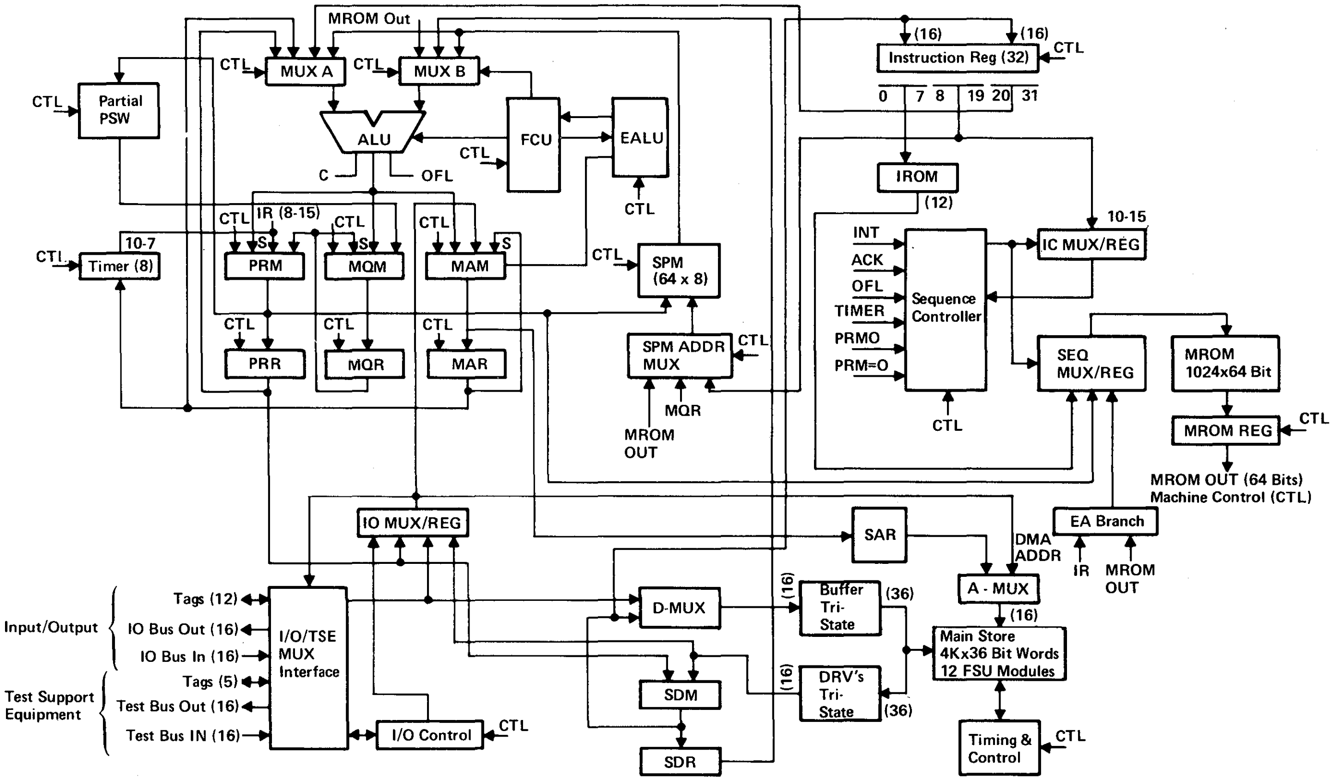


Figure 1—HTC block diagram

PARTITIONING CONSIDERATIONS

One of the most difficult and important decisions made in the early design phase of the simulation involved partitioning of the simulation into models. More concisely, the question was "what function went in what model?"

The first approach was to partition the models along functional lines. For example: main memory, arithmetic logic unit, and sequencer control unit were three of the partitions decided upon. The advantage of this type of partitioning was ease of description of model function, and ease of interconnection. Since the inputs and outputs of these models were already described in the functional block diagrams, the only remaining problem was determining how the outputs were generated. Once the algorithms for generation of outputs were obtained from the hardware designers these models could be developed easily.

This type of partitioning did have one major disadvantage. Since some functional blocks consisted of several hardware chips, low level logic for all chips in a functional block were required to be ready before any of these chips could be integrated into the simulation. Since this could seriously delay the simulation of some low level chips, a modification of the original partitioning approach was necessary. Hence, those functional blocks which were composed of more than one chip were broken down into an equal number of high level models.

There were three types of exceptions to the splitting of functional partitions. The first of these exceptions was

```
MODEL OF PRODUCT REMAINDER REGISTER (PRR)

DCL     PRMUX CHAR(16);                         /* INPUT FROM PRMUX */
DCL     CLOCK CHAR(1);                          /* CLOCK LINE FOR PRR*/
DCL     SELECT CHAR(1);                         /* REGISTER SELECT LINE*/
DCL     PRR CHAR(16);                           /* REGISTER OUTPUT */

DCL     OLDCLOCK CHAR(1) INIT('U');             /* OLD CLOCK STATE*/
DCL     TEMP CHAR (16);                         /* TEMPORARY STORAGE*/

/*      START OF PRR MODEL        */

IF CLOCK = OLD CLOCK THEN RETURN                /* CHECK FOR CLOCK STATE CHANGE */
OLDCLOCK=CLOCK;                                 /* IF CHANGE THEN SAVE NEW STATE*/
IF(CLOCK¬ = '0')|(SELECT¬='1') THEN RETURN;     /* RETURN UNLESS SELECTED AND CLOCKED*/
TEMP = PRMUX;                                   /* SAVE PRMUX LINE STATE*/
CALL WAIT(XX);                                  /* DELAY XX ns */
PRR = TEMP;                                     /* OUT PUT REGISTER CONTENTS */
END;
```

Figure 2—PRR model

```
MODEL OF INSTRUCTION READ ONLY MEMORY (IROM)

DCL     ADDRESS CHAR(8);                        /* IROM ADDRESS LINES*/
DCL     IROMOUT CHAR(16);                       /* IROM 16 OUTPUT LINES */

DCL     OLDIN CHAR(8) INIT('UUUUUUUU);
DCL     IROM (256) FIXED (16) EXTERNAL;         /* IROM MEMORY*/
DCL     TEMP CHAR (16)                          /* TEMP STORAGE*/
DCL     I  FIXED BIN (31);
DCL     ISUM FIXED BIN (31);

/*   START IROM MODEL  */

IF ADDRESS = OLDIN THEN RETURN;                 /* CHECK FOR ADDRESS CHANGE*/
OLDIN = ADDRESS;                                /* SAVE OLD ADDRESS STATE */
PACK (ADDRESS, 8, ISUB);                        /* CONVERT ADDRESS TO BINARY FOR ARRAY*/
CALL WAIT (100);                                /* DELAY 100 NS –MAX IROM DELAY*/
I = IROM (ISUB+1);                              /* ADDRESS DATA FROM ARRAY */
UNPACK (I,16,TEMP);                             /* CONVERT BINARY TO CHARACTER*/
IROMOUT = TEMP;                                 /* OUTPUT IROM DATA */
END;
```

Figure 3—IROM model

TABLE 1—HTC High Level Models

Model		Inputs	Outputs	Statements
MROM	- Micro-program Read Only Memory	12	64	40
MREG	- MROM Register	65	64	27
IROM	- Instruction Read Only Memory	8	16	17
FCU	- Function Control Unit	21	15	226
ARCH	- Architecture Unit	29	16	210
AREG	- Arch Register	12	9	85
MUXA	- ALU A-side MUX	63	16	70
MUXB	- ALU B-side MUX	50	16	80
ALU	- Arithmetic Logic Unit	41	19	114
EALU	- Extended ALU	8	2	84
PRM	- Product Remainder Mux	56	20	111
MAM	- Memory Address Mux	55	20	122
MOM	- Multiply Quotient Mux	52	16	61
PRR	- Product Remainder Register	18	16	19
MAR	- Memory Address Register	18	16	18
MQR	- Multiply Quotient Register	18	16	19
SPM	- Scratch Pad Memory	23	16	56
SS	- Single Shot	1	1	13
IO	- Input Output	24	17	400
SIL	- Storage Interface Logic	41	47	375
AMUX	- Address Mux	34	28	79
SPRO	- Storage Protect	9	2	59
ADR	- Storage Data Register	42	32	67
DMA	- Direct Memory Access	72	34	117
MSP	- Memory Support	26	42	79
TRID	- Tri-State Drivers	38	18	62
PAR	- Parity	18	1	55
MMEM	- Main Memory Array	80	36	151
IREG	- Instruction Register	19	32	68
SPAM	- Scratch Pad Address Mux	16	4	31
TIME	- Timing	24	8	126
SCU	- Sequencer Control Unit	85	22	220
SEQR	- Sequencer Register	14	12	20
ICR	- Iteration Control Register	8	6	20
DCOD	- Decoder	5	8	46
	Total Inputs/Outputs/Statements	1105	707	3347

made for functional models composed of vendor off-the-shelf devices. Since these devices, such as read only memories, were purchased and required no detailed design verification, these functional models were not subdivided.

The next exception was made for functional blocks composed of multiple chips of the same type. For example, the sixteen bit wide ALU was composed of eight 2-bit wide chips. Since all of these chips were identical, their low level logic would all be ready simultaneously, and could be integrated into the simulation together to replace the one high level functional model.

The third exception was made for functional blocks which were known to consist of multiple chips but whose chip design had not progressed enough to identify the inputs, outputs, and functional partitioning of these chips. The major example of this type of functional block was main memory. The approach taken for this type of block was to begin with one high level model for main memory.

```
RIPPLE ROUTINE

DCL    LIMIT CONSTANT(6) FIXED;
DCL    RIPPLE (LIMIT) FIXED(16);
DCL    NEXT FIXED(15) INIT(0)
DCL    TEMPB FIXED (32);
DCL    DELAY CONSTANT(60) FIXED;
/*     OUTPUT TEMPB   */

IF NEXT > 0
       THEN IF RIPPLE (NEXT) = TEMPB
          THEN RETURN;
          ELSE;
       ELSE IF RIPPLE(1) = TEMPB THEN RETURN;
IF NEXT = LIMIT
THEN DO;
       MESSAGE 'RIPPLE TABLE EXCEEDED – DATA LOST';
       RETURN;
       END;
NEXT = NEXT + 1;
RIPPLE (NEXT) = TEMPB;
CALL WAIT (DELAY);
TEMPB = RIPPLE (1);
NEXT = NEXT - 1;
DO     I = 1 TO NEXT;
       RIPPLE (I) = RIPPLE (I + 1);
       END;
```

Figure 4—Ripple routine

As the design progressed, high level models were written for the components of main memory as sufficient information became available, but no model was integrated into the simulation until all high level models which were to replace the original memory model were ready. At that time they were all integrated into the simulation. The corresponding low level chips could then be integrated as they became available.

MODELING CONSIDERATIONS

Once the partitioning into models and the assignment of inputs and outputs to these models were complete, the actual coding of the high level models was relatively easy. Using the hardware designers functional description and his estimate of the time delay involved in generation of the outputs, the operation of the model was coded in the high level language.

Although each model was different there were a few areas common to almost all models. First, a check was made at the start of each model to insure that at least one input had changed from the last time the model was executed. This eliminated redundant execution of the model, thus saving computer time. Next the outputs were generated using the current set of inputs. Quite often this generation took the form of a Boolean equation expressed in PL/I statements. When all outputs had been computed a test was made to determine whether any output had changed. If not, nothing more was done.

Once the new outputs had been computed and had been found to be different than the last computed outputs, these outputs could still not be set on the output lines until the specified hardware delay had occurred. In order to delay outputs and also show any ripple that might take place due to new inputs, a ripple routine (see Figure 4) was used. When a new and different output was computed it was put in the next available position in an array and a wait was executed for the proper delay. After the delay the first element of the array was output, all other elements were moved down one slot and the next available entry pointer was decremented. This technique allowed several outputs to be computed before the delay time of the first element had expired and also allowed all of the outputs to be actually output at their correct time.

TESTING

The first stage in testing the simulation was testing of the individual models as they were coded. Aside from the normal syntax and keypunch error debug, test cases were run using generated input patterns to insure that the model being tested generated a predictable output at the proper time from the known input pattern.

The next phase of testing involved groups of related models such as main memory, data flow, etc. Again, test cases were run with generated input patterns for this particular group of models. This phase of testing made apparent any interconnection, polarity, or timing problems within the group of models.

The final test phase involved the entire group of high level models. While the first two phases had been primarily involved with debug of the models, this phase was primarily involved in debug of the micro-code and the hardware design of the HTC.

In order to make the simulator as flexible as possible, all memory models—Instruction Read Only Memory (IROM), Micro-code Read Only Memory (MROM), and Main Memory (MMEM)—loaded their simulated contents from S/360 OS data sets at execution time. This technique allowed the micro-programmers to place the latest versions of IROM and MROM in a predefined data set and their latest versions were always included in the simulation. In order to change the contents of main memory, the job control language could merely be modified to point to a different data set.

The basic philosophy used in final testing of the micro-code and final design of HTC was developed around a self test approach. Since the HTC instruction set was the S/360 set, it seemed apparent that the best debug programs would be S/360 diagnostics. Thus, the memory contents used in driving the simulation were various S/360 diagnostic programs designed to test the particular areas of the HTC under test at that time. Since these diagnostics were designed to hang in a one instruction loop when a failure was detected, the simulation was designed to detect this occurrence and initiate appropriate action to alert the user. The simulation would print an error message, dump simulated memory, force the program counter to the next sequential instruction, and then continue. When, however, a user defined number of errors were detected or a user specified number of simulated time units was reached, the simulation was terminated.

Simulation output consisted of several types of information. First, the simulator generated certain error and information messages such as the previously mentioned one instruction loop message. In addition, all critical registers

```
                                       PROGRAM COUNTER: 0214,   INSTRUCTION: LA
CURRENT SIMULATED TIME=0000011732NS
SPMO              = 0000000000000000
IRG               = 01000001000100000UUUUUUUUUUUUUUU
SDR               = 0100000100010000
CPSAR             = 0000001000010110
PRM               = 0000000000000010
ICREG             = 000000
CURRENT SIMULATED TIME=0000011800NS
PRR               = 0000000000000010
MAR               = 0000001000010100
MQR               = 0000001000010110
MROMIN            = U00100101111
MREGIN            = 0000000101011111011000010000100011100000010001001100000000000100
CURRENT SIMULATED TIME=0000012132NS
SPMO              = 0000001000010110
IRG               = 01000001000100000000000000000011
SDR               = 0000000000000011
CPSAR             = 0000001000010110
PRM               = 0000001000011000
ICREG             = 000000
CURRENT SIMULATED TIME=0000012200NS
PRR               = 0000000000000010
MAR               = 0000001000010100
MQR               = 0000001000010110
MROMIN            = 000100110000
MREGIN            = 1100001001110110010000010000100011100000010001001101000000000000
CURRENT SIMULATED TIME=0000012532NS
SPMO              = 0000000000000000
IRG               = 01000001000100000000000000000011
SDR               = 0000000000000011
CPSAR             = 0000001000010110
PRM               = 0000000000000011
ICREG             = 000000
CURRENT SIMULATED TIME=0000012600NS
PRR               = 0000000000000010
MAR               = 0000001000010100
MQR               = 0000001000010110
MROMIN            = 000100110100
MREGIN            = 1000001001111110010000010000100011100000010001001111000000000000
CURRENT SIMULATED TIME=0000012932-000
SPMO              = 0000000000000000
```

Figure 5—Trace of microcode execution

such as the program counter, storage data register, instruction register, etc., were printed once in every micro cycle. This allowed a trace of micro-code execution and showed the results at key points. Figure 5 is an example of this type of output. Finally, in case of problems in which the error was not apparent, the user could ask for complete timing charts of any input or output lines in the simulation. These charts printed the value of the line and the current simulated time each time the state of that line changed. Using these print charts, almost any error could be traced to its source.

CONCLUSIONS

Several benefits were derived from the simulation approach. Probably the most significant benefit was the ability to discover costly errors before a single piece of hardware was built. The errors discovered were the type typically found during a hardware debug. They included timing, logic, polarity, micro-code, and design errors. An additional benefit was the ease with which the causes of these errors could be found because lines could be examined via simulation that could not be physically probed on an LSI chip.

The objectives of the simulation, as stated previously, were verification of the computer design and micro-code. How well the verification was accomplished could not be answered until the HTC power on debug began. The time from initial power on of the HTC until debug was complete was slightly less than one month with no design problems encountered. HTC debug was accomplished in this amount of time because the micro-code and hardware debug actually occurred in the previous months via software simulation.

Synchronous microcomputer system for on-board missile guidance and control

by FRANK J. LANGLEY and JOSEPH J. COONEY
Raytheon Company Missile Systems Div.
Bedford, Massachusetts

STATEMENT OF THE PROBLEM

The design, development and production of missiles to cover a range of presently defined missions with the capability of being upgraded to accommodate changing threat situations and advancing technology without major redesign, stresses the need for more modular guidance and control electronics possessing both physical and electrical flexibility features at lowest cost.

Although programmable digital techniques have been shown to offer improved performance and greater flexibility than traditional hardwired analog implementations, the direct substitution of a single, real-time, general-purpose, digital computer to perform the on-board guidance and control task does not provide an optimum solution in many cases. While throughput can be readily satisfied for all missile types with a single, standard, mini-class computer, an excessive performance margin results in the low and medium performance missiles, and, the centralization of a single computing unit presents form-factor incompatibilities across the range of missiles together with a poor electrical interface. In addition to the latter deficiencies, peculiar to the missile application, the design, assembly and checkout of major missile sections/functions (e.g., seeker, guidance, autopilot, attitude reference, umbilical/command-link interface, warhead fuzing) as completely operational modules is not possible with a central computer design approach. However, the recent advent of low-cost, medium-speed microcomputer component sets in high-density N-channel metal-oxide-semiconductor (MOS) technology provides a form of programmable digital hardware, complete with support software, which allows more optimum system design tailoring at any stage in the life cycle without cost-penalty.

REQUIREMENTS

These are presented in two parts commencing with an overview of missile types, their distinguishing characteristics and the nature of the guidance and control task viewed as a sampled-data system. This is then followed by a breakdown of the guidance and control functions for each missile type with the resulting computer loads expressed in terms of the respective memory capacities and operation execution rates for specific instruction mixes.

Missile types and characteristics

Of the four major missile families, surface to surface, surface to air, air to ground and air to air, (A-A) the latter group was used as a basis for detailed requirements analysis, since missiles in this family cover a wide spectrum both in physical form factors and electrical performance characteristics. Data derived from these analyses were then augmented with the results of other studies in digital surface to air missiles and highly modular, multi-mission, air to ground weapons.

At one end of the air to air missile spectrum are the small light-weight, short-range, single mission weapons designed typically for tail-chase engagements using passive infrared homing seekers. Mid-spectrum types include the larger, medium-range, high-performance missiles equipped with semi-active radar target seekers, and, at the upper end of the missile range, multi-mode missiles constitute the most sophisticated air to air weapons, incorporating semi-active and active radar seekers supplemented with command guidance from the launch aircraft. Figure 1 illustrates form factor variations typical of the three main types of A-A missiles just described. The functional partitioning and arrangement of major component parts in tandem along the longitudinal axis becomes a design prerequisite in each case due to the rigid constraints of the fundamental air vehicle design.

Figure 2 shows the relative compatibility of a single computer versus a multiple microcomputer system for on-board guidance and control observing the previously outlined physical constraints. The improvements in subsystem autonomy and interface are clearly evident in the multiple microcomputer configuration.

Sampled data guidance and control system

From a digital sampled-data mechanization viewpoint an on-board guidance and control system appears as shown in Figure 3.

TABLE I—On-Board Missile Guidance & Control Functions

MAJOR FUNCTION	NO.	SUB FUNCTION	IR MISSILE		RADAR MISSILE		MULTIMODE MISSILE	
			MIN	MAX	MIN	MAX	MIN	MAX
TRACK & STABILIZATION	S1	BASIC T&S	●	●	●	●	●	●
		BASIC T&S WITH ESTIMATION			●	●	●	●
	S2	RADOME COMPENSATION			●	●	●	●
	S3	HEAD AIMING (INITIALIZING)	●	●	●	●	●	●
	S4	GYRO & TORQUE DISTURBANCE COMPENSATION					●	●
	S5	LINEAR 9-STATE FEEDBACK CONTROL						●
STATE ESTIMATION	E5	LEAD-LAG NOISE FILTER	●					
	E1	FIXED GAIN FILTERS		●				
	E2	SWITCHED GAIN FILTERS			●			
	E3	VARIABLE GAIN FILTERS (DECOUPLED KALMAN)				●	●	
	E4	VARIABLE GAIN FILTERS (COUPLED KALMAN)						●
GUIDANCE	G1	PROPORTIONAL NAVIGATION	●	●	●			
	G2	4-STATE LAW, RANGE DESENSITIZED				●	●	●
AUTOPILOT	A1	BASIC A/P	●	●	●	●	●	●
	A2	STRUCTURAL FILTERS & FIN MIXING			●	●	●	●
	A5	INDUCED ROLL REDUCTION				●	●	●
	A6	BAND-SWITCHED GAINS	●	●	●			
	A9 A10	GAIN DETERMINATION WITH AERO ESTIMATES (α)				●		
	A13 A14	GAIN DETERMINATION WITH AERO ESTIMATES (FIN CROSS COUPLING)					●	
	A15 A16	GAIN DETERMINATION WITH AERO ESTIMATES (SINGLE PANEL MODEL)						●
ATTITUDE REFERENCE	11	ATTITUDE DETERMINATION				●	●	●
	12	VELOCITY DETERMINATION				●	●	●
	13	POSITION DETERMINATION				●	●	●
	14	ANGLE OF ATTACK DETERMINATION				●	●	●
	15	AERO PARAMETER ESTIMATION				●	●	●
	16	MASS & BALANCE ESTIMATION				●	●	●

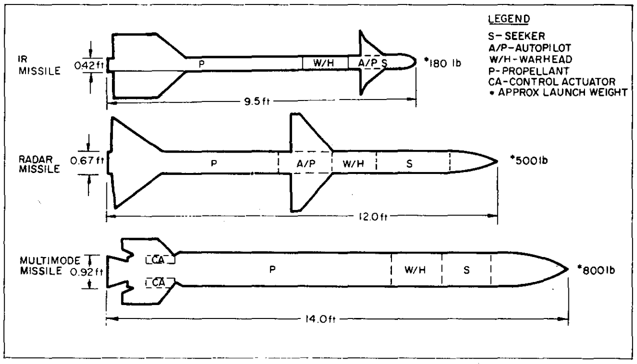


Figure 1—Air to air missile types and form factors

Switches are shown with associated sampling rates at the inputs and outputs of all major functional blocks. The basic stability augmentation loop, formed by the autopilot, control actuators, control surfaces/fins, aerodynamics and instrumentation (body gyros and accelerometers), requires sampling rates of 125 to 500 Hz depending on missile type and performance. An inner position feedback is shown as autonomous with the control actuator servos. Steering inputs to the missile flight system can originate from various on-board sources: target homing seekers, (e.g., radar, infrared, electro-optical), mid-course navigation sensors (e.g., inertial, Tacan, Tercom, Loran), and/or from the launch vehicle via a command guidance radio frequency link. In all cases, these steering inputs require low sampling rates ranging from 10 to 20 Hz. The on-board target seeker shown in Figure 3 as a typical example, employs a target sensor mounted on a gimballed platform which in turn is stabilized against missile body motion by a higher speed control loop (125-500 Hz) similar to that of the flight system. The target-sensor boresight error vector (ϵ), derived in the signal processor, becomes the common low frequency steering command to both the seeker head/platform control subsystem and the estimation and guidance section for translation into acceleration commands (N_c) to the autopilot.

Hence, in terms of system timing, digital on-board guidance and control systems are characterized by two distinct sampling/update intervals, viz: 2 to 8 ms for body-motion-related functions and 50 to 100 ms for steering or pointing commands. Further, since steering command rates are very slow and can originate from many different asynchronous sources, the flight system becomes the main system "clock" to which all system steering inputs must be synchronized.

TABLE II—Computer Requirements vs Microcomputer Performance—IR Types

IR TYPES

REQUIREMENTS			MICROCOMPUTER THROUGHPUT	
MAJOR FUNCTION	MEMORY	*THROUGHPUT (MIX)	BIPOLAR	** N-MOS
HEAD CONTROL	120	25.0 (80/20)	300.0	20.0
ESTIMATION & GUIDANCE	280	5.0 (70/30)	250.0	15.0
AUTOPILOT	150	15.0 (92/8)	480.0	42.0

*THROUGHPUT: (MIX) THOUSANDS OF COMPUTER OPERATIONS PER SECOND WITH OPERATION MIX AS PERCENTAGE SHORT/LONG OPERATIONS, e.g. ADD TYPES/ MULTIPLY TYPES

ASSUMING WORST-CASE MULTIPLY TIME (230 μsec)

TABLE III—Computer Requirements vs Microcomputer Performance—Radar Types

RADAR TYPES

REQUIREMENTS			MICROCOMPUTER THROUGHPUT	
MAJOR FUNCTION	MEMORY	THROUGHPUT (MIX)	BIPOLAR	N-MOS
HEAD CONTROL	700	60.0 (82/18)	320.0	22.0
ESTIMATION & GUIDANCE	1000	20.0 (70/30)	230.0	15.0
AUTOPILOT	520	100.0 (84/16)	340.0	24.0
ATTITUDE REFERENCE	700	40.0 (82/18)	320.0	22.0

Computer functions and loads

The specific characteristics of each major functional block shown in the generalized guidance and control system diagram of Figure 3 vary in degree of complexity according to missile type and performance. Table I gives a breakdown of the major functions with their respective applications across the range of air to air missiles. Minimum and maximum performance categories are also indicated for each class. A detailed discussion of these functions and their impact on missile performance is given in two recent publications (References 1 and 2) and therefore will not be repeated in this paper. In summary, the essential differences between the six complements of functions listed in the table lie in the degree of compensation of seeker errors (radome refraction, gyro torque disturbances); the availability of missile/target relative motion data, including acceleration, due to Kalman estimation techniques; the use of a more optimal guidance law based on the availability of improved target performance data; the use of an inertial reference system to provide missile performance data, and the resulting design of an autopilot which is both adaptive to the aerodynamic environment and compensates for body bending and fin blanking effects.

Computer loads resulting from the above functions are summarized in Tables II, III and IV for each missile type, respectively. Memory requirements include word space for both programs and data bases. These loads were derived chiefly from algorithms and program listings used in current and previous in-house digital missile development programs which employed a 16-bit minicomputer in single central computer system configurations. The remaining new algorithms were derived and sized in the same man-

TABLE IV—Computer Requirements vs Microcomputer Performance—Multi-Mode Types

MUTLI-MODE TYPES

REQUIREMENTS			MICROCOMPUTER THROUGHPUT	
MAJOR FUNCTION	MEMORY	THROUGHPUT (MIX)	BIPOLAR	N-MOS
HEAD CONTROL	900	160.0 (85/15)	350.0	25.0
ESTIMATION & GUIDANCE	3000	60.0 (73/27)	240.0	15.0
AUTOPILOT	2500	170.0 (86/14)	370.	27.0
ATTITUDE REFERENCE	670	220.0 (83/17)	330.0	23.0

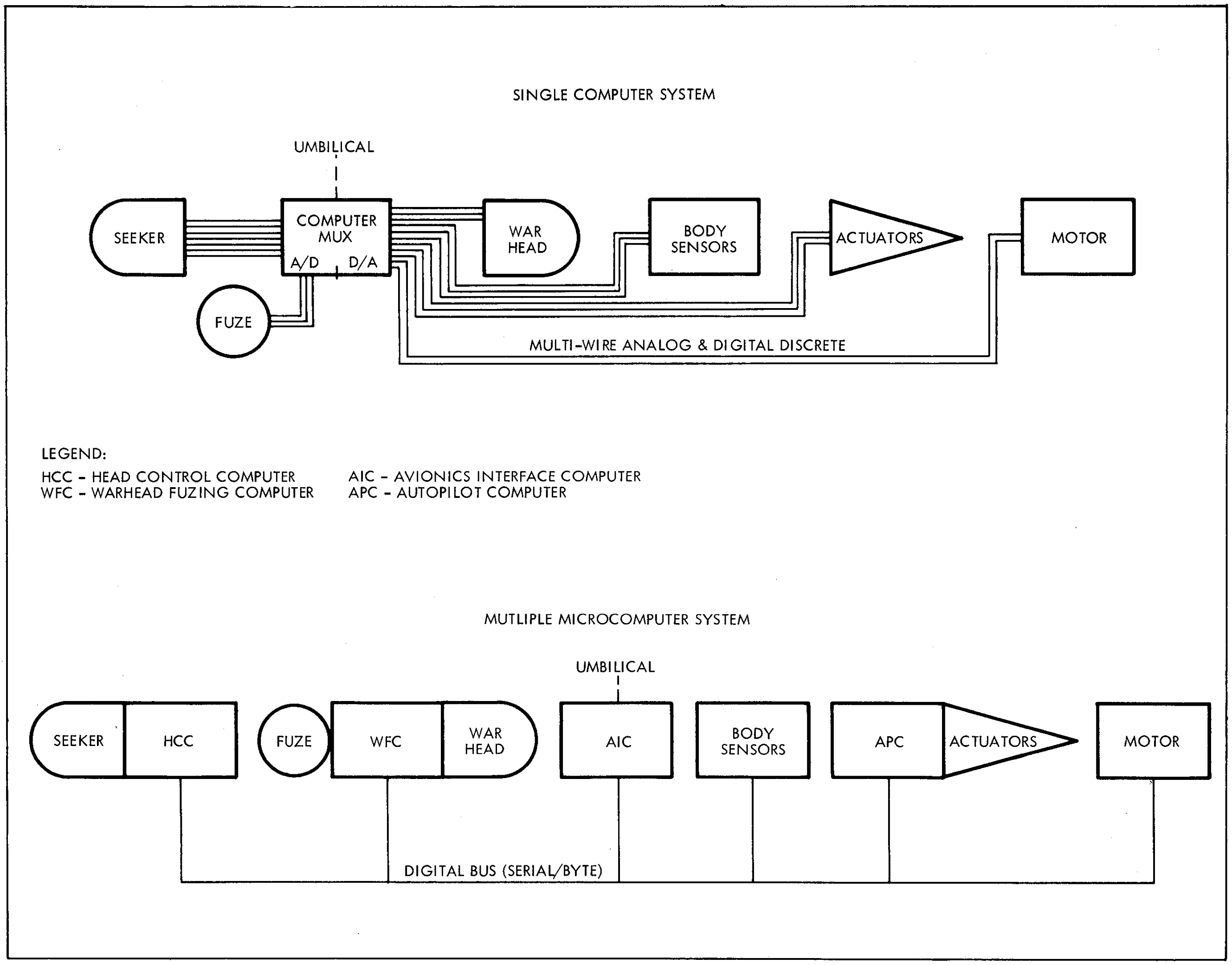


Figure 2—Single computer vs multiple microcomputer systems within missile form factor and assembly constraints

ner, assuming the same simple machine architecture and instruction set. Throughput rates are based on the update rates previously discussed.

Implicit in the above computer sizing data is the partitioning of the total guidance and control task by major function, instead of by control channel (pitch, roll and yaw) as in conventional analog implementations. Partitioning by major function stems from the single computer approach to the problem, and although the resulting computer loads appear to be within the performance capabilities of the lower-cost medium-speed, N-channel MOS microcomputers for the small IR seeker missile, the throughput requirements become excessive for the other two types. On the other hand, bipolar microcomputers with increased power consumption and parts count exceed the IR missile throughput requirements by an order of magnitude and the performance required for the second type of missile by 3:1.

Repartitioning the system by control channel, however, considerably improves the desirable match between the available performance of the lower cost MOS computer hardware and the throughput requirements of the higher performance missile. Table V illustrates this point for the radar missile, and similar gains in cost, power and parts count could be expected for the multi-mode class using the

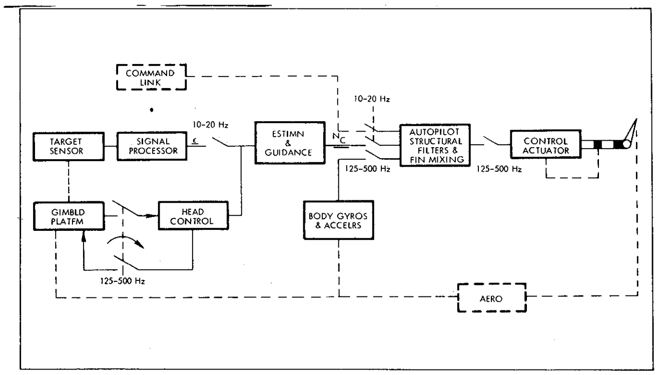


Figure 3—Digital missile guidance and control system

same microcomputer as a common, programmable function unit/cell, (Reference 3).

MICROCOMPUTERS

Of the many different types of microcomputers currently produced by semiconductor manufacturers, two common architectures are evident. One is characterized by a complete 4- or 8-bit central processing unit (CPU) packaged on a single large-scale-integrated-circuit chip, "CPU-on-a-chip", using metal-oxide-semiconductor (MOS) technology. The other form of microcomputer uses a 2- or 4-bit register arithmetic and logic unit (RALU) module, in lower density bipolar device technology, which forms a common element or slice to build larger RALUs up to 32 bits in length. To form a complete CPU requires additional control memory modules and addressing circuits, enabling the design of a microprogram-controlled CPU with virtually any desired instruction set in contrast to the hardwired, fixed repertoire imbedded in the CPU-on-a-chip version. Both types use a common input output bus, to/from the CPU, to which is coupled main memory and external device control modules, allowing direct memory access for data transfers without a programming burden.

Table VI compares the parts complements of two 8-bit microcomputers, CPU—vs RALU-on-a-chip types, each incorporating 2048-words of program read-only memory (ROM) and 256-words of data read/write memory (RAM). Reductions in parts count and power consumption are typically 4:1 for the higher density MOS computers together with approximately a 3:1 reduction in cost, compared to bipolar versions. CPU-on-a-chip types are slower (2μs register-register add) than their RALU-on-a-chip counterparts ($<1\mu$s register-register add), and currently include no hardware multiply or divide instructions in their otherwise comprehensive instruction sets. Software multiply times can vary between 230μs (16×8) to 33μs (8×8), depending on the degree of optimization possible for a given application. The longer period was assumed in determining the N-MOS computer throughputs given in Tables II, III, IV, and V.

In summary, since the leading CPU-on-a-chip microcomputers are offered as complete component sets with the lowest parts count, together with the advantage of full support software, they constitute the most effective product of the two available microcomputer architectures. Further, the low cost, size, weight and power advantages of CPU-on-a-chip microcomputers constitute significant incentives to reevaluate the missile guidance and control problem in the light of the availability and performance of such a versatile computing tool.

TABLE V—Partitioning By Control Channel
RADAR TYPES

	THROUGHPUT REQUIREMENTS			MICROCOMPUTER THROUGHPUT
MAJOR FUNCTION	PITCH	ROLL	YAW	N-MOS
HEAD CONTROL	30.0	-	30.0	22.0
ESTIMATION & GUIDANCE	10.0	-	10.0	15.0
AUTOPILOT	30.0	30.0	30.0	24.0

TABLE VI—Available Microcomputer Components

CPU-ON-A-CHIP TYPES	RALU-ON-A-CHIP TYPES
(HIGH-DENSITY P & N-MOS)	(MEDIUM-DENSITY BIPOLAR & MOS)
CPU: 4 & 8 BIT (1) RAM: 256 x 4 (2) ROM 2048 x 8 (1)	RALU: 2 & 4 BIT SLICE (2) CROM: CONTROL UNIT COMPLETE (12) RAM: 256 x 1 (8) ROM: 256 x 4 (16)
CLOCK, I/O LATCHES/DRIVERS (6)	CLOCK, I/O LATCHES/DRIVERS (2)
8 BIT 10 DIPS, 5 WATTS, $200.00 COMPUTER:WITH FULL SUPPORT SOFTWARE	8 BIT: 40 DIPS, 20 WATTS, $600.00 COMPUTER: NO SUPPORT SOFTWARE AVAILABLE

SYSTEM DESIGN APPROACH

As outlined earlier in this paper, the guidance and control of a missile using a digital system entails the repetitive sampling of body motion sensors and the computing and outputting of fin position commands at rates of between 125 and 500 Hz to satisfy the basic stability requirements of the air vehicle. Maneuvering commands from target/navigation sensors, or remote sources via command data links, require sampling and inputting to the flight system at significantly lower rates of between 10 and 20 Hz. Hence, any asynchronism between the latter data inputs and the higher speed stability loop inputs can be accommodated by temporary buffering and delayed accessing in synchronism with the next flight control cycle.

The initiation and use of so-called "modes" is a convenient way of describing quasi-static missile performance changes which are required in response to pre-determined values of target/navigation sensor or command guidance data, the net result of which is typically: the use of data from an alternate sensor, (e.g., radar to IR); a change in sensor sensitivity; and/or changes in autopilot gains and the gearing or response of the system to steering command inputs. Such performance changes can be accomplished through normal limit checks on sensor input data at the 10-20 Hz rate, followed by changes, if necessary to stored missile performance constants regularly accessed at the 125 to 500 Hz rate.

From the preceding system design considerations it can be seen that a minimum form of general-purpose computer would satisfy the system requirements. Specifically, a computer possessing a direct-memory-access (DMA) input-output channel for asynchronous data inputs, and a simple cyclic program, without real-time executive and interrupts, to perform the repetitive processing task. Further, to keep hardware at a minimum, a simple 8-bit byte-organized machine would suffice, provided that the required throughput could be maintained with multiple-byte, higher-precision operations. In other words, the processing task lies within the province of a simple microcomputer.

Figure 4 shows the missile guidance and control system of Figure 3 reconfigured with three "CPU-on-a-chip" type

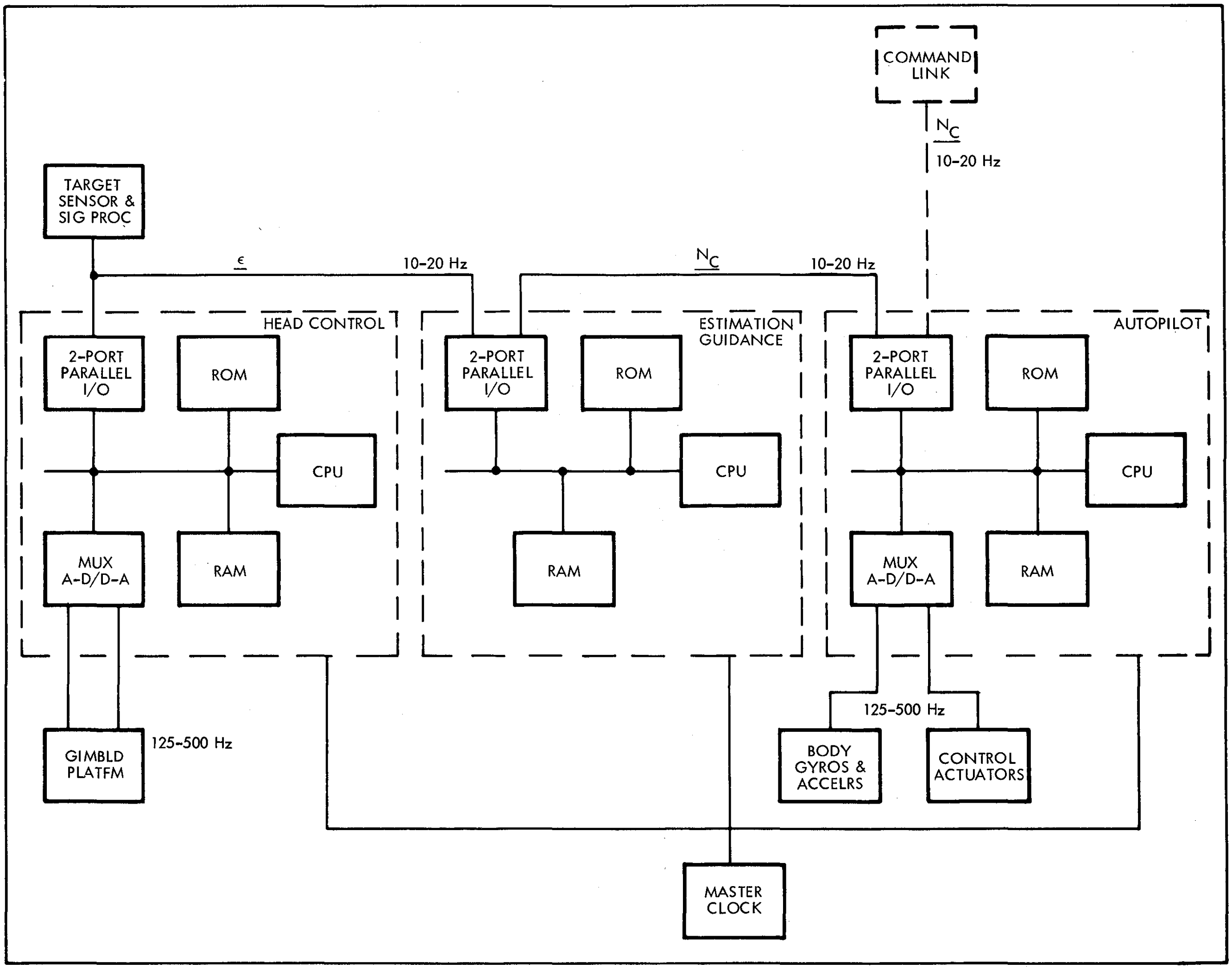


Figure 4—Synchronous microcomputer guidance and control system

microcomputer sets programmed and dedicated to: seeker head control, (tracking and stabilization), state estimation and guidance, autopilot and fin command generation, respectively. This form of system partitioning using medium-speed, MOS microcomputers is compatible with the IR and lower performance radar-type missiles.

TABLE VII—Performance, Parts Count, Power & Cost

COMPONENT	PARAMETER	ONE MINI (16 BIT)	FOUR MICROS (8 BIT EA)	
			6701	8080
CPU	THROUGHPUT	600.0	300.0	25.0 TO 120.0
	PARTS	150	60	25
	POWER	24.0	24.0	14.0
RAM	CAPACITY	1024 x 16	256 x 8 EA	256 x 8 EA
	PARTS	16	32	8
	POWER	8.0	14.0	1.2
ROM	CAPACITY	4096 x 40	2048 x 8 EA	2048 x 8 EA
	PARTS	160	64	4
	POWER	98.0	39.0	2.0
TOTAL	PARTS	326	156	37
	POWER	130.0	77.0	17.2
	COST	$4500	$2500	$700

THROUGHPUT: KOPS
POWER: SECONDARY WATTS
COST: COMMERCIAL (FOR DIRECT COMPARISON)

Input-output transfer rates between the three computers are slow and amenable to a single-wire serial digital interface of the type currently advocated for avionics system integration, (i.e., tri-service DAIS).

High speed I/O interfaces occur only between the computers which are physically colocated with their external devices i.e., gyros, accelerometers, position pickoffs, torque motor drivers and control actuators.

System timing is governed by a master clock synchronized to the carrier frequency of the alternating current pickoffs thereby enabling peak sampling of the carrier and the elimination of hardware demodulator units. Two major timing templates are used to control and synchronize the sampling and processing cycles of each computer. One template employs minor intervals of 2-8 ms duration with steering inputs interleaved at 50 to 100 ms intervals for the head control and autopilot computers, while the other uses the longer interval alone to control the estimation and guidance computer. Upon completion of a data processing program each computer idles pending the synchronous transfer of processed data and inputting of new data samples. Further details of subsystem timing are described in the following section, based on a developmental model of the head control section recently implemented with an N-MOS microcomputer set.

Table VII compares throughput, parts, power and cost of four 8-bit microcomputers, in bipolar and N-MOS technology, each with 256 words RAM and 2048 words ROM versus one 16-bit minicomputer with 1024 words RAM and 4096 words control ROM. From this comparison, it can be seen that extensive replication of higher-density MOS computer hardware is possible beyond four computers (i.e., up to 16) before the cost, parts and power consumption of the higher-speed equivalents is reached.

MICROCOMPUTER SEEKER HEAD CONTROL - DEVELOPMENT MODEL

The following describes the practical implementation of the head control function using a leading commercial N-MOS microcomputer. This computer is scheduled for second-source production to military specifications in mid-1975.

The system (Figure 5), incorporates an Intel 8080 8-bit CPU module with associated ROM, RAM and I/O modules, (8 LSIC packages total), together with digitally controlled analog input and output multiplexers, to time share the single A-D and D-A convertor modules. Subsystem timing is governed by the synchronization logic. The need for phase-sensitive demodulators is avoided by sampling gyro outputs at the peak of the reference waveform. This is accomplished without the use of interrupts by synchronizing the system to the reference frequency. Figure 6 shows the timing of input and output operations with respect to the gyro reference frequency. From the timing diagram it can be seen that the computer is idle 44 percent of the time, enabling system cost to be further reduced by using this spare capacity to implement such gyro error compensation and signal conversion functions described in the following paragraphs.

Gyros are g-sensitive devices and therefore their output is a function of both angular motion and linear acceleration. Although g-sensitivity can be reduced by careful balancing and compensation, this results in an expensive component. Alternatively by permitting the gyro to have a higher but known g-sensitivity, approximately a 50 percent cost reduction can be realized. The accuracy equivalent to that of a low g-sensitivity gyro can be achieved in software by correcting the gyro output as a function of the linear acceleration to which it is exposed. Since most missiles measure linear acceleration in airframe coordinates for guidance and control purposes, this information can be used after coordinate transformation to provide the acceleration component along the gyro input axis. The gyro output signal can then be corrected in the head control computer as a function of the latter acceleration, resulting in reduced system cost.

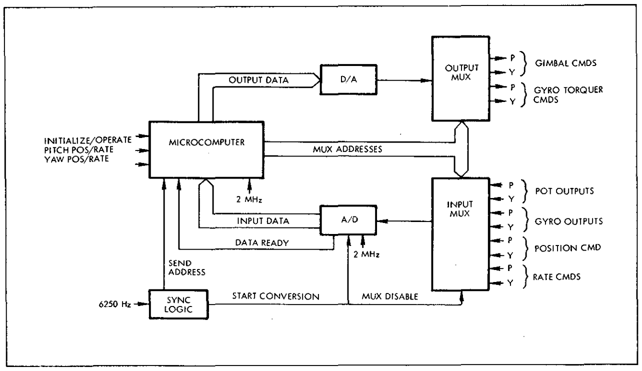


Figure 5—Microcomputer head control subsystem

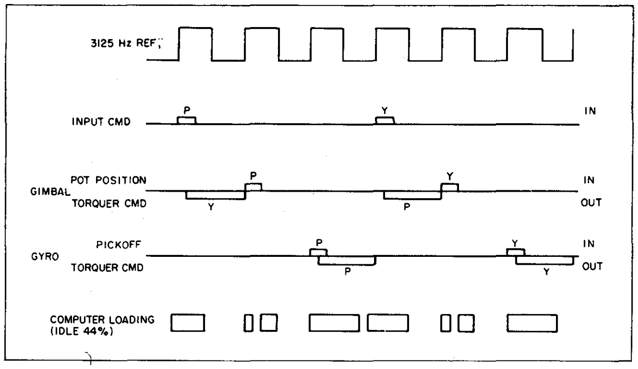


Figure 6—Microcomputer head control subsystem timing

With the distribution of the guidance and control task among several computers, both the individual throughput and analog to digital (A-D) conversion rates are reduced compared to a single computer configuration performing high-speed, time-division-multiplexed operations. Low-speed A-D conversion provides the option of performing the successive approximation operation by microcomputer subroutine and replacing the A-D converter with a simple comparator. The unknown voltage is then compared to the output of the lower cost D-A convertor unit with the result used to set the latter to a new value successively, until the equivalent digital value is determined.

In addition to the above hardware cost reduction techniques, incremental improvements in overall missile performance can be achieved in the head control computer by the substitution and/or addition of program modules listed in Table I.

CONCLUSIONS

From the requirements analyses, product surveys, design trade-offs and experimental work reported in this paper, a synchronous, multiple microcomputer guidance and control system, of the form described, achieves a high degree of compatibility with the unique design constraints of the various missile families.

The availability of high-density, large-scale-integrated-circuit, N-MOS microcomputer component sets allows the federation of separable guidance and control functions in compliance with the physical partitioning and modular construction of missiles without the size, weight, power and cost penalties formerly associated with the replication of computer hardware. Further, significant improvements in missile performance, design flexibility and logistic support are provided through the use of a common set of standard high-volume-production computer components, which in turn enables a standard digital interface to be established between both the major missile sections and the avionics of the launch vehicle.

Lastly, the federated computer system described permits the partial conversion of existing missile guidance and control systems to obtain the desired performance improvements without the trauma of a major redesign and consequently at low-risk and at incremental funding rates.

ACKNOWLEDGMENTS

The authors wish to acknowledge the guidance and support of Messrs. John J. Long, Dr. Bruce A. Hall, and Walter V. Trainor, in the preparation and critique of this paper, also Mr. David S. Siegel, Office of Naval Research, who initiated and supported the earlier work (Reference 1) which stimulated the follow-on efforts described herein.

REFERENCES

1. *Modular Digital Missile Guidance System Study* Raytheon Co., Missile Systems Division, Final Report June 30, 1974 Contract No. N00014-74-C-0056, AD-784 969/8GA.
2. Terzian, J. and W. Trainor, "Digital Missile Technology," *1974 ADPA/A1AA Tactical Missile Conference,* Canoga Park, Calif., May 1-3, 1974.
3. Langley, F. J., "A Universal Function Unit for Avionic and Missile Applications," *NAECON 1971,* Dayton, Ohio, May 17-19, 1971.

A new fourth generation of hybrid computer systems

by ROBERT M. HOWE
The University of Michigan
Ann Arbor, Michigan

and

ALDRIC SAUCIER
Hqtrs. US Army Materiel Command
Alexandria, Virginia

INTRODUCTION

It has long been recognized that hybrid computer systems are more cost effective than digital computers in solving many types of dynamic problems. This indeed has been the reason for the sizable number of hybrid computer systems currently in operation. The one to two order of magnitude speed advantage which hybrid computers enjoy over digital computers in solving differential equations is due to the speed of the analog subsystem. This high speed results from the high bandwidth (up to 1 megahertz) of the analog components and the parallel configuration of the analog mechanization.

Despite this speed advantage, hybrid computers have not enjoyed anywhere near the widespread use which one might have predicted. There are several reasons for the relatively lower popularity of the hybrid computer vis-a-vis the digital computer. These include the greater difficulty in programming hybrid computers, their limited accessibility to multiple users, expensive and limited problem-storage capability because of the analog patchboards, and relatively slower problem changeover characteristics.

Recently the U.S. Army Materiel Command sponsored a study program by several of the hybrid computer manufacturers to determine the characteristics of a next-generation hybrid computer system which will overcome the existing disadvantages of hybrid computation as listed above. The resulting AHCS (Advanced Hybrid Computer System) achieves this by replacing the analog patchboards by a digitally-controlled solidstate switch matrix and by use of a higher-level simulation language compiler which assigns analog components to problem variables and parameters. The switch matrix allows complete digital storage of hybrid problems and problem turnaround in milliseconds. This in turn makes hybrid computer sharing through remote terminals a practical reality. The CSSL-type input language makes the programming as simple as digital programming using an identical language and permits implementation of automatic scaling of the analog subsystem.

In this paper we summarize the results of the study by one of the contractors of the Army Materiel Command, namely the Applied Dynamics Computer Division of the Reliance Electric Company. Prior to the study there has been considerable experience with a prototype electronically-patched hybrid computer system which has been operating for three years in the University of Michigan Simulation Center.[1] This computer consists of an Applied Dynamics AD Four analog subsystem interfaced to a Digital Equipment PDP 9 digital subsystem, as shown in Figure 1. This system, which includes a matrix of 768 switches for interconnecting 8 analog integrators, 6 multipliers, and 32 coefficient devices, allows analog circuits for solving differential equations to be electronically patched and completely set up in less than 20 milliseconds under PDP 9 control. Five remote graphics terminals are used to "time-share" the system. Each terminal user is able to request the solution of any analog problem stored in the digital computer. Problem parameters are entered interactively from the terminal, and high-speed solutions are displayed on a storage-type CRT as shown in Figure 2. Since problem setup takes only 20 milliseconds and typical problem run times are under 100 milliseconds, each terminal ties up the hybrid computer for such a short time that the mean response time from solution request to solution display at a given terminal is a fraction of a second. A simulation-language compiler has also been developed for the system.[2] From CSSL-type input statements the compiler selects analog components and generates the switch code necessary to patch the problem.

This time-shared or, perhaps more properly, time-sliced auto-patch system with terminals at the University of Michigan (Applied Dynamics has installed three similar systems in the United States and Europe) provided useful background for the US Army Materiel Command (AMC) Advanced Hybrid Computer System (hereafter called the AHCS). Let us now turn to the results of the study.

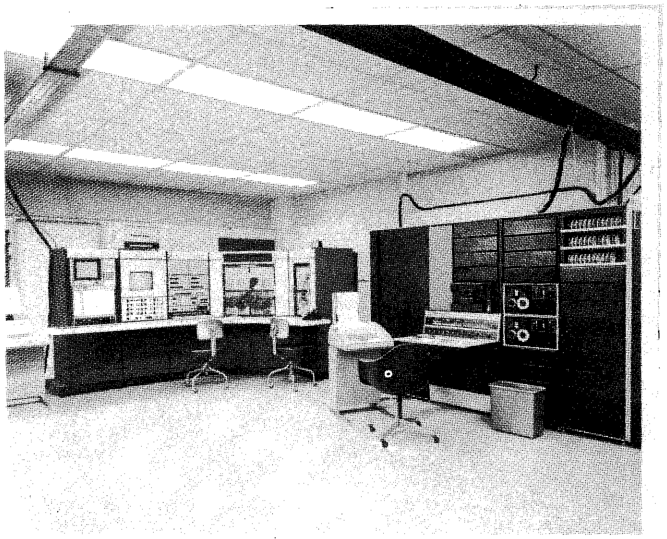


Figure 1—Hybrid computer system at the University of Michigan Simulation Center

OVERALL AHCS CONFIGURATION

Figure 3 shows a block diagram of one proposed version of the AHCS. At the bottom of the diagram is the modular analog computer with electronic patching which will be described in subsequent sections. Across the top of the figure are the user terminals for remote access. These can either be standard digital terminals with graphics capability (either storage or refresh CRT's) or can be specially-designed terminals for the AHCS. The user terminals, as well as the maintenance terminal and a large digital computer, are all tied to the AHCS through the communications controller. Note that the user terminals can receive analog signals from the AHCS (dashed lines in Figure 3) although it is our feeling that digital communication, even for solution displays at the user terminals, will in general be preferred. Even at telephone-line communication rates it will only take about a second for a typical solution display to be generated. Since the AHCS will compute solutions much faster than this, it is necessary to convert these solutions to digital form, store them temporarily, and transmit them through the communications controller to the user terminals for display. This is the purpose of the two analog to digital converter (ADC) systems, micro-computers, and associated mass storage devices shown in Figure 3.

Also shown in the figure are two medium-sized digital computers connected through interfaces to the analog computer and to the communication controller. These two digital computers share common peripheral devices (disk, line printer, tape drives, etc.) and are used to compile, allocate, schedule, and perform other tasks connected with set-up, check-out, and problem runs on the AHCS system. They are also used for performing digital computations associated with actual problems, such as digital function generation, equation integration, etc. Provision of two such digital computers as well as two micro-computers for solution storage and display adds redundancy and hence increases AHCS reliability. The large digital computer shown in Figure 3 can also be used for compiling. Let us now turn to a description of the most unique subsystem in the AHCS, namely the electronically patched modular analog computer.

THE SWITCH MATRIX

A real key to the AHCS is a low-cost, high-reliability switch matrix to replace the analog patchboard. The 768-switch matrix used in the prototype autopatch system currently in operation is implemented using discrete Field Ef-

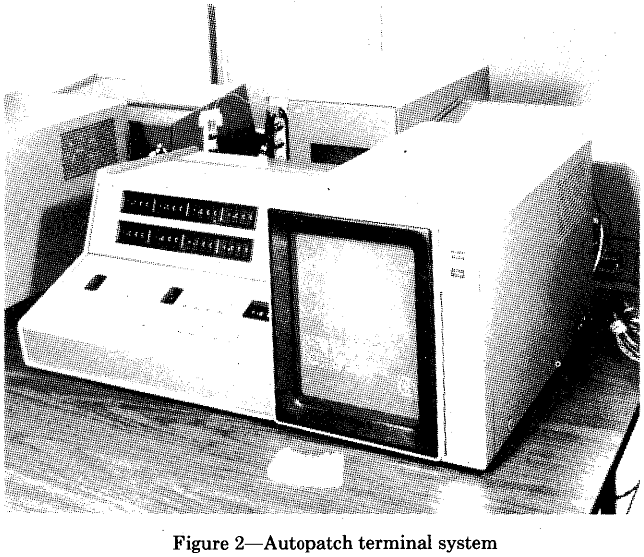


Figure 2—Autopatch terminal system

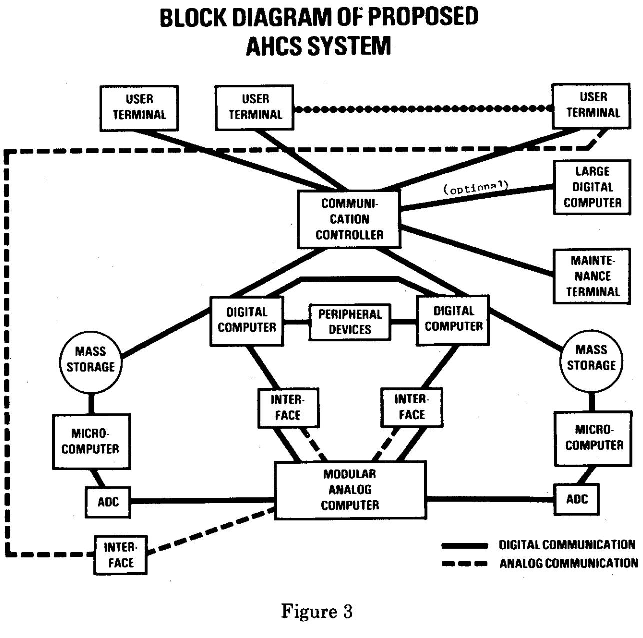


Figure 3

fect Transistors (FET) switches and driving-circuit components, although integrated circuit registers are employed for storing the switch-closure information for a given patch configuration. In the AHCS it is proposed to use Complementary Metal Oxide Semiconductor (CMOS) LSI circuitry for the analog switches and the switch-setting registers. It appears that it will be feasible to mount a 16 × 8 voltage-switch matrix plus associated digital registers on a single chip. The effect of finite switch "on" resistance (approximately 500 ohms) will be eliminated by terminating each of the 8 matrix outputs with voltage-following amplifiers. The 16 matrix inputs come from low-impedance operational amplifier outputs. Leakage current from "off" switches as well as capacitive coupling across "off" switches will cause negligible cross talk because of the low-impedance voltage sources driving the switch inputs. Normal range of the input and output voltages for the switch matrix will be + 10 volts. It is estimated that using such a specially-developed 16 × 8 LSI switch chip it would be practical to mount up to 5000 switches on a single printed-circuit card.

TABLE I—Modular Categories for the AHCS

Type of Module	Module Contents
Amplifier	8 Summer-Integrators
	32 MDAC's
	12 RDAC's
	8 Multipliers
	2 Hard Limiters
Function Generator	16 Two Variable 16×8 DFG's
Resolver	4 Resolvers
Logic	16 Comparators
	16 Parallel-Logic Microprocessors
	Counters
	Shift Registers
	Flip Flops
	One-Shots
Miscellaneous	Log DFG's
	Time-Delays
	Programmable Nonlinear Functions

MODULAR ORGANIZATION OF THE ANALOG SUBSYSTEM

Every study of automatic patching in recent years has recognized the necessity of organizing the parallel processor into modules in order to reduce the number of switches required to allow interconnection of components. In defining the modular breakdown and the switch matrix within and between modules, one must take into account the types of problems which will be solved, the component and switch hardware cost and performance requirements, and the complexity of software needed for the compiler which converts the simulation language source program into component assignments and a scaled circuit. Because of the experience with the prototype autopatch AD-Four hybrid system at The University of Michigan Simulation Center, we feel that we have been able to perform the various tradeoffs needed to come up with the reasonable hardware-software compromise for configuration of a practical autopatch system.

The system utilizes five general categories of modules, as illustrated in Table I. The largest number of modules in a typical installation fall into the amplifier-module category. As can be seen in the figure each amplifier module contains 8 summer-integrator amplifiers, 32 MDAC's (multiplying digital-to-analog converters), 8 multipliers, and 2 hard limiters. The function-generator module contains 16 two-variable digitally-set analog diode function generators. The resolver module contains 4 resolvers, each capable of either forward or inverse resolution. The logic module includes 16 comparators along with 16 parallel-logic microprocessors. The miscellaneous module is simply that, a module type which contains components such as fixed function generators (e.g., log dfg's), time-delay elements, special programmable nonlinear functions, etc. All of the modules can be interconnected with the autopatch switch matrix used for both analog and logic signals, as will be described. Also, the modules can be connected to the digital computer or computers in the AHCS, as well as to external analog and digital trunks. References 9 and 10 give a more detailed description of each of these modules.

TYPICAL ANALOG EQUIPMENT COMPLEMENT

Since the analog subsystem of the AHCS involves parallel mechanization, it is clear that there must be enough analog and parallel logic components to solve the largest problem envisioned for the computer. A typical system might have 40 modules total, consisting of 24 amplifier modules, 5 resolver modules, 5 function generator modules, 2 miscellaneous modules, and 4 logic modules. Such a system would include the following impressive component count:

192 summer integrators
192 multipliers
768 MDAC's
288 RDAC's
48 hard limiters
20 resolvers (80 additional multipliers plus 20 sine-cosine generators)
80 two-variable function generators
64 comparators
64 parallel-logic microprocessors (logic function generators)
64 16-bit up counters
16 sequencer/timer/counter units

The total number of switches needed to implement the automatic patching of the above system is over 100,000. Although this sounds like a prohibitively large switch count, the actual number of discrete components, using LSI technology (see The Switch Matrix Section), would be approximately the same as the number of components used in the 768-switch matrix currently employed in the AD-Four autopatch system. The existing system has proved to be quite reliable in operation over the past three years in The University of Michigan Simulation Center.

A typical large aerospace dynamics problem, the simulation in six degrees-of-freedom of an aircraft, has been used along with other example problems to determine the adequacy of the proposed switch matrix and compiler algorithm. For the aircraft simulation problem, which required 8 functions of three variables, 7 functions of two variables, and 4 functions of one variable along with 7 resolutions, the following modules were needed: 5 amplifier modules, 3 function generator modules, 2 resolver modules.

A SPEED COMPARISON WITH ALL-DIGITAL COMPUTATION

In order to provide a quantitative comparison between the speed capability of the AHCS and conventional serial digital computers, the number of digital operations (additions, multiplications, divisions, equivalences, etc.) corresponding to the various analog module operations were estimated.[3] In this comparison it was assumed that a fourth-order Runge-Kutta integration formula was used for the digital mechanization, so that each problem function needed to be evaluated four times per integration step. It was concluded on this basis that 992 digital operations per integrator step are needed to match the capability of an amplifier module, 800 per step to match the resolver module, and 1984 per step to match the function generator module. To match the performance of the amplifier, resolver, and function generator modules in the Typical Analog Equipment Complement Section would require 37728 digital operations per integration step.

If we further assume that approximately 20 integration steps per cycle of the highest problem frequency are needed for reasonable accuracy[4] and the highest problem frequency which the AHCS can handle with reasonable accuracy is 1 Khz, then 37728 (20) (1000) = 754,560,000 digital operations per second are needed to match the AHCS speed capability. This is some three orders of magnitude faster than all but the most powerful existing digital machines.

Of course, the above comparison is not quite fair since it assumes that all the components within each AHCS Module are used, which in general they are not. To get a more reasonable comparison, we estimated the number of digital operations per integrator step for the six-degree-of-freedom aircraft simulation described at the end of the Typical Analog Equipment Complement Section, and found the number to be 5456. This compares with 12512 operations per step based on use of all components in the 5 amplifier, 2 resolver, and 3 function generator modules needed for AHCS mechanization of the same problem. Thus to match a 1 khz AHCS problem frequency capability in simulating six-degree-of-freedom aircraft equations would require 5456 (20) (1000) = 109, 120,000 digital operations per second. Again, however, this is two to three orders of magnitude faster than existing serial digital machines.

OTHER SYSTEM FEATURES

In this paper we have concentrated on a description of the analog subsystem modules, electronic switch matrix, and computing speed capability of the AHCS. These result in the performance capabilities which give the system two to three orders of magnitude potential cost advantage over conventional digital systems in solving dynamic problems. However, to realize these savings the AHCS must overcome the other traditional disadvantages of analog and hybrid computation. In this section we discuss briefly how this will be done.

Operation through remote terminals

In order to improve computer accessibility and utilization the AHCS will be operated through remote terminals on a time-sliced basis. As described in the Switch Matrix Section, digital data transmission between the central AHCS computer and the terminals will in general be used, thus allowing terminal tie-in through standard telephone lines over unlimited distances. Program input and editing will be accomplished at the terminal as well as graphical display of solutions. Because of the anticipated short response time (one second or less in many cases), AHCS operation through the terminals will be extremely interactive. Some terminals may have local computation capability for off-line source program preparation and editing. Hard-copy capability in various forms (CRT display copy, x-y plotter, multiple-channel oscillograph) will also be available at the terminals.

Simulation language input

As stated earlier in the paper, the basic source program language for the AHCS will be a CSSL-type simulation language. Not only is such a language extremely easy to learn, but, very importantly, it can be used as a common language for both the AHCS and an all-digital check solution. This has already been accomplished for hand-patched hybrid systems.[5,6] Having a digital check solution not only is useful as a check on the AHCS solution but is also needed to provide maximum variable ranges to establish scaling information for the AHCS compiler. An alternate method for automatic scaling involves an interactive scheme which stops the analog solution and rescales the appropriate variable following a variable overrange.

It is proposed that other input languages (block diagrams, transfer function, etc.) would be overlays on the standard CSSL AHCS language. Conventional hybrid computation (combined use of analog and digital systems while computing a solution) would also be a compiler mode of operation. As indicated earlier, however, the AHCS is designed to maximize the all-analog mechanization capability in order to exploit system computing speed to the utmost.

Dynamic relocation of modules

As discussed in the Speed Comparison With All-Digital Computation Section, it is clear that the analog subsystem must be large enough to handle the biggest problem required to be solved on the AHCS. Obviously, many problems will be much smaller than this. For this reason the AHCS should have the capability of solving more than one problem at a time by simultaneously allocating different modules to different problems. The proposed switch and component numbering code makes this extremely easy to do without any requirement for recompiling when shifting a given problem over to different modules. Thus the AHCS will not only time-share in series but also in parallel.

Continuous preventative maintenance

One of the advantages of an electronically patched hybrid computer is the ease with which a diagnostic program can be implemented to check the proper functioning of all the analog components and switches, and the speed with which such a program can be executed. This has already been done successfully for the AD-Four autopatch system and will be an integral part of the AHCS. In fact, with the dynamic relocation of modules described in the previous paragraph it is completely feasible to run diagnostic checks on any modules while other modules are used for solving problems. On a continual basis each module can be taken successively out of the AHCS inventory and exercised with a complete diagnostic check, say every several minutes. Fault readout and interpretation can be accomplished through a standard terminal located at the AHCS site and assigned to maintenance operations. The built-in redundancy which such a scheme implies should provide the AHCS with a very high overall reliability.

CONCLUSIONS

In this paper we have described a proposed fourth generation Advanced Hybrid Computer System (AHCS) with electronic patching and computing speed capability two to three orders of magnitude faster than conventional digital computers in solving dynamic problems. This computational speed plus fast problem turnaround (a fraction of a second) makes practical a sequential sharing of the AHCS through remote terminals. Also, dynamic relocation of analog subsystem modules makes parallel sharing possible, as well as continual maintenance checking. A CSSL-type input language along with automatic scaling capability allows users with no computer programming expertise to learn very quickly the use of the system. Full graphics capability at the terminals along with quick computer access and short solution times will allow a high-degree of man-machine interaction in problem solving. It is estimated that cost of such a system will be comparable to existing hybrid computer systems with the same component complement. This is because of the utilization of CMOS LSI circuits and other IC technology in the AHCS. Since the proposed system overcomes the disadvantages of the present hybrid computers (difficult programming, manual scaling, poor turnaround time, etc.) it is felt that the AHCS, shared through remote terminals, will represent a major breakthrough in cost-effectiveness for solving dynamic problems.

PROPOSED ARMY MATERIEL COMMAND COMPUTER NETWORK

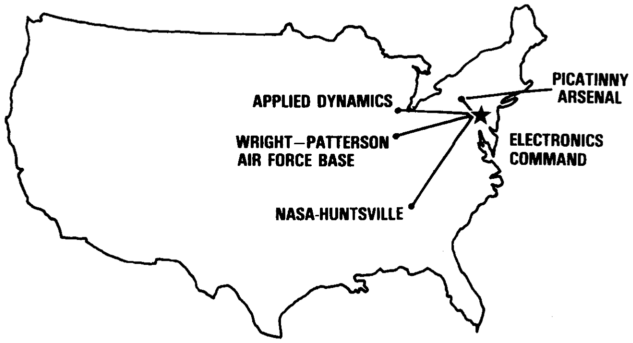


Figure 4

Figure 4 shows a proposed Army Materiel Command computer network using both pure-digital and also hybrid computer systems in the 1978-79 time frame. Such a network has the following operational advantages: (1) access to large-scale digital and hybrid computers by users who do not have on-site machines; (2) access to specialized programs or technology available at particular computing centers; (3) load-leveling among computer centers; (4) decreased computing costs; (5) increased computing power. The network also has the following managerial advantages: (1) greater ease and precision in identifying total computing workload requirements; (2) larger and more stable base from which to make workload projections; (3) ability to add capability to the network as a whole rather than at each individual installation; (4) computer power can be added in increments which more closely match requirement; (5) responsiveness to "political" pressure to utilize fully the available resources by sharing computers.

In summary, the distribution of computing power of a large scientific and engineering computer network, including both pure-digital and hybrid computers, will provide both parallel and serial compabilities completely programmable from remote terminals. This in turn provides the user with increased computer power and speed, and at less cost.

REFERENCES

1. Howe, R. M., R. A. Moran, and T. D. Berge, "Time Sharing of Hybrid Computers Using Electronic Patching," *1970 Fall Joint Computer Conf., AFIPS Conf. Proc.,* Vol. 37, AFIPS Press, Montvale, N.J. 1970, pp. 377-386. See also *Simulation,* Vol. 15, pp. 105-112, Sept. 1970.
2. Howe, R. M. and R. B. Hollstein, "Time-Shared Hybrid Computers: A New Concept in Computer-Aided Design," *Proc. IEEE,* Vol. 50, No. 1, pp. 71-77, Jan., 1972.
3. *A Performance Comparison of All Digital Computation and the Advanced Hybrid Computer System,* Internal Report, Applied Dynamics Div., Reliance Electric Co., Ann Arbor, Michigan, May 1974.
4. Gilbert, E. G., "Dynamic Error Analysis of Digital and Combined Analog-Digital Computer Systems," *Simulation,* Vol. 6, No. 4, pp. 241-257, April 1966.
5. Rigas, H. B. and D. J. Coombs, "Patch: Analog Computer Patching from a Digital Simulation Language," *IEEE Trans. Comput.,* Vol. C-20, pp. 1140-1146, October 1971.
6. Rook, R. E., "Actran: An Analogue/Hybrid Compiler," *Proc. 6th AICA Cong.* (Munich, Germany, Aug. 31 - Sept. 4, 1970).
7. *Advanced Hybrid Computer Systems Technology Report;* Project Leader, Aldric Saucier, HQTS US Army Materiel Command, June 1973.
8. Plan for an AMC Scientific and Engineering Computer Network Chairman, Dr. Ronald P. Uhlig and David L. Grobstein, HQTS US Army Materiel Command, August 1972.
9. *Advanced Hybrid Computer Systems Final Report,* Army Contract DAAG39-74-C-0076, by Applied Dynamics Computer Division, Reliance Electric Company, Ann Arbor, Michigan, May 1974.
10. Howe, R. M., "The Next Generation of Hybrid Computer System," *National Electronic Conference,* October 1974.
11. McKechnie, R. M., "Generalized Vehicle Dynamics Program for Interactive Hybrid Computer Graphics," *Computer Aided Design for Engineering Conf.,* June 1973.

Design and application of electronically programmable LSI arrays

by D. HAMPEL
RCA
Somerville, New Jersey

and

R. L. BARRON and D. CLEVELAND
Adaptronics, Inc.
McLean, Virginia

INTRODUCTION

It has been shown that arrays of calculating elements, providing various functions of their input variables, can be used as general-purpose signal processors.[1-9] Such elements could, in general, operate on binary, analog, or numerical inputs. Networks or arrays composed of elements in each of these domains have advantages in particular applications. The numerical processing element and its use in programmable arrays is the subject of this paper.

The major potentials of such arrays lie in their ability to perform reliable high-speed processing, and their ability to be used in adaptive control systems.[10-12]

The function repertoire of the basic element has been defined as a family containing both linear and nonlinear multinomial expressions. The particular function of the element at a given time, its inter-connectivity within an array, and the coefficients or weights of its multinomial terms are all controllable. These features provide an array of such elements with a high degree of flexibility, and allows such an array to be alternately programmed to solve a large variety of problems. Also, such an array can be "trained" in that it can rapidly accept different sets of weights and connection commands until it represents a transformation acceptable as a solution to a given problem. Although limited versions of such arrays have been built in hardware they have been simulated, for the most part, in software. This has often confined the application of such arrays to off-line processing or experimental work. With the objective of efficient array realization an analysis of programmable array hardware requirements has been made, tradeoffs in its implementation have been completed, and an optimum architecture has been determined. Based on state-of-the-art LSI technology, projections as to array performance were derived. This derivation led, in turn, to the definition of a custom CMOS/SOS LSI circuit which would serve as a key ingredient in the processor of the element. This circuit and its use in programmable arrays will be described. Examples of array applications in aerospace are presented.

PROCESSING ELEMENT AND ARRAY STRUCTURE

A basic element is depicted in Figure 1(a); the figure shows its major control and input and output signals. Such elements are, in general, structured in multilayered nets, as shown in Figure 1(b). The interconnection control, which is part of each element, but which appears functionally between successive layers of elements, has inputs which can route the output signals of any one element to the desired input of an element in the next layer. Each element and interconnect switch in the array has sufficient memory for storing the function type it is to compute, the necessary weights or coefficients of its function, and the interconnect data. Hence, this programmable array can be considered as a distributed logic-memory processor capable of rapid reconfiguration and calculation of complex transformations.

Before describing the element requirements and architecture, the three possible configurations of programmable arrays will be explained; these configurations are shown in Figure 2. In Figure 2(a), a net of dimension $j \times k$ is shown fully populated, with each element containing an m-bit store for necessary function and control. This configuration can process (or transform) $j/2$ inputs at a time and can be operated in a pipeline mode so that k sets of data are simultaneously operated upon in different layers.

In Figure 2(b), a single layer of elements can be multiplexed to simulate a whole net. The memory of each element must now have km bits to provide the necessary control as the processor of the element acts, in turn, to realize each of the k layers. For any single problem the speed is the same as in Figure 2(a), but pipelining cannot be done. In Figure 2(c) a single processing element with jkm bits of storage can be used to process inputs within a layer and then successive layers. Provisions must also be made for storage of intermediate results in Figures 2(b) and (c); approach (c) will have 1/j the speed of (b).

It is envisioned that arrays of up to 256 elements (or equivalent) could provide the processing capacity for a

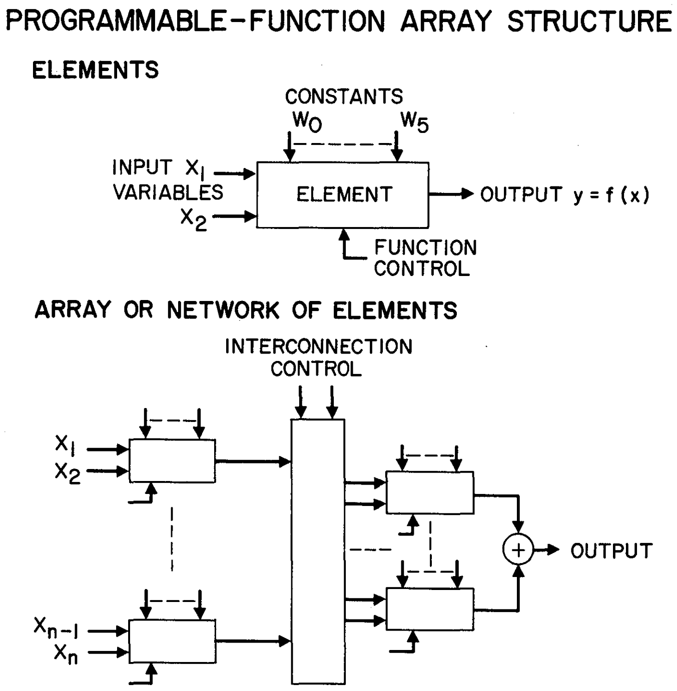


Figure 1—Programmable-function array structure

vast majority of applications. Hence, the design, using either the configuration in Figure 2(b) or (c), will have the capacity for memory expansion to simulate nets of up to 256 elements. These arrays will, in general, be loaded and controlled by a computer, but may have input/output signal interfaces as well, as shown in Figure 3. As such, the programmable arrays can be considered as firmware, reconfigurable upon command, for high-speed processing.

Element definition and description

The basic numerical processing element of the programmable array will have the capability to perform the following functions:

$P1\ y = W_0 + W_1 W_1$
$P2\ y = W_0 + W_1X_1 + W_2X_2 + W_3X_1X_2$
$P3\ y = W_0 + W_1X_1 + W_2X_2 + W_3X_1X_2 + W_4X_1^2 + W_5X_2^2$
$P4\ y = W_0X_2 - X_1X_2^2$
$P5\ y = W_0 + W_1X_1 + W_2X_2 + W_3X_3 + W_4X_4 + W_5X_5$

Functions $P1$, $P2$, and $P3$ are useful for general-purpose linear and nonlinear hyper-surface calculations or transformations. $P4$ was provided for realizing division by a recursion formula. $P5$, a linear combination of five variables, is ideally suited for digital filters and linear transforms, such as the FFT. Each of the five functions could be realized by specifying only one element. For example, $P1$ could be used twice to realize $P2$, etc. However, substantially greater speed and efficiency is achieved by providing a micro-programmed control to optimally evaluate each function.

The element will be given the capability of operating on 32-bit floating-point values with a 24-bit mantissa and 8-bit exponent. It will also be capable of operating on fixed-point values of up to 24-bits with increased speed and lower package count.

The basic element is shown in Figure 4; it consists of an arithmetic processor, an element controller and random-access memories (RAM's).

The arithmetic processor performs additions, subtractions, and multiplications of the input variables and weights to compute the various polynomials. These operations may be either fixed or floating point, depending on the requirements of the array to be synthesized. Use of an iteration scheme permits division to be simulated with several elements.

The element controller consists of a read-only memory (ROM) containing microinstructions for implementation of the desired polynomial repertoire and associated logic to translate these microinstructions into control signals for the arithmetic processor and address information for the RAM's. The RAM's contain the polynomial function selection and elment interconnection information, poly-

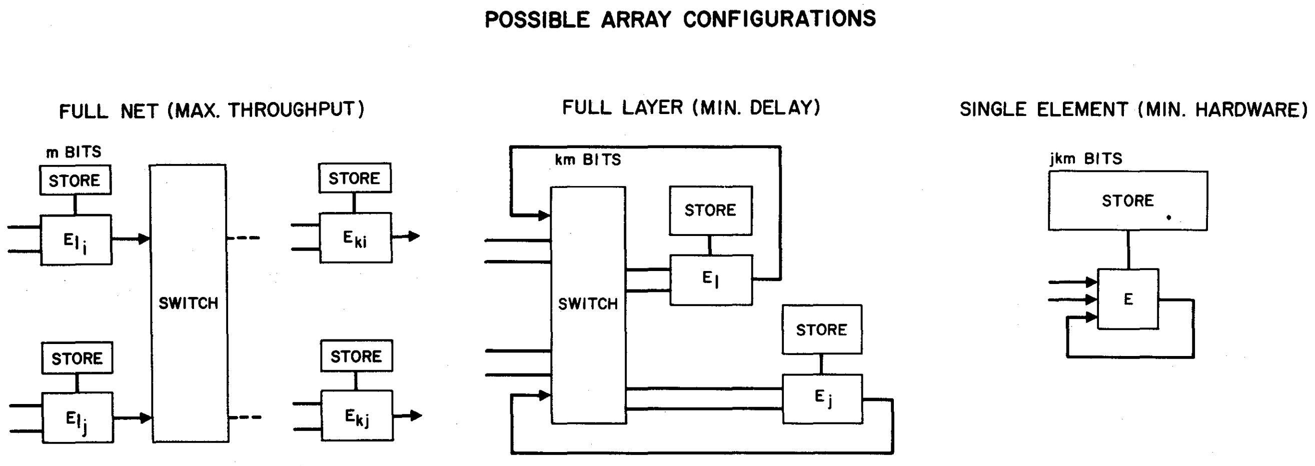


Figure 2—Possible array configurations

nomial weights, and input/output variable storage for the element. Element interconnection is accomplished by specifying the RAM address of each required input variable. Output variables are stored in sequential RAM locations. If more than one element is used to simulate an array, the input variables might be stored in the RAM's associated with another element. In this case, one or more inter-element buses are provided to exchange data between elements. Since any element can access any previously generated output in this manner, complete interconnection flexibility is accomplished. An intra-element bus provides flexible data routing within the element. In operation, a computer "programs" each element by loading the proper polynominal select codes, input addresses, and polynomial weights into the RAM's via the intra-element bus. The computer then loads initial input variables into the variable memory via the intra-element bus, and provides an "execute" signal to the element controller.

Each element controller will sequentially step through the previously trained portions of its associated memory, performing the desired element operations and storing the results in the variable memory. This execute phase is complete when the elements detect a polynomial select code equivalent to a halt instruction. At this point the element controllers will output a "ready" signal to the computer. Upon receipt of the ready signal, the computer retrieves the output data by providing output addresses directly to the element controllers and receiving the output data via the intra-element buses.

Design approaches

Given the preceding operational constraints, what is the best technology and architecture to provide a good balance between speed and complexity or cost? The major hardware consideration impacting these criteria is the multiplier implementation. Two major organization types of elements were investigated, one using a high-speed parallel multiplier (with and without pipelining) and the other a high-speed serial/parallel multiplier. Estimates as to total chip count and speed were made for each approach and for variations within each approach. Available LSI and MSI IC's are considered along with key LSI multiplier chips which would have to be developed for the overall element realization.

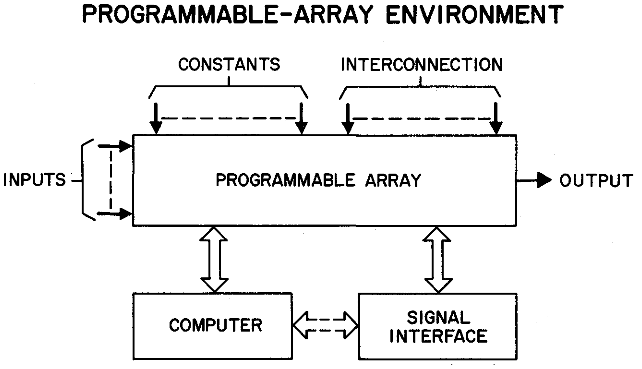


Figure 3—Programmable array environment

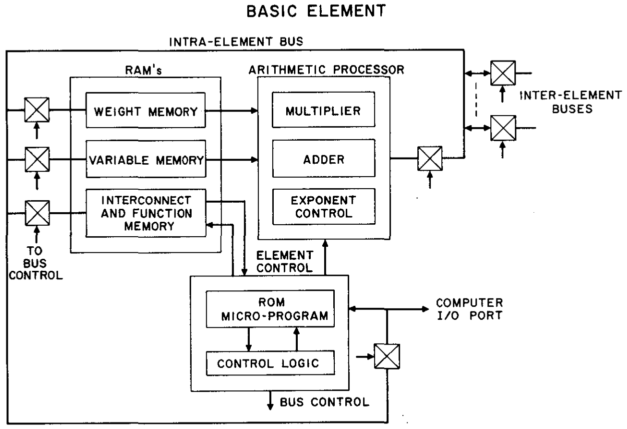


Figure 4—Basic element

The necessary IC's to make the desired programmable arrays a viable processing scheme can be realized in a variety of emerging technologies. Very-high-speed LSI multipliers have been developed or are in development in both bipolar and CMOS/SOS form.[13,14] The choice of IC technology for each section of the element is dictated by availability and performance of existing circuits (particularly RAM's), as well as the best realization of any required new LSI development. These factors led to the definition of a CMOS/SOS serial/parallel multiplier chip compatible in speed and logic with associated control and memory. The serial nature of the design capitalizes on the "on-chip advantages" of SOS, and the high packing density of CMOS/SOS (compared to bipolar implementations) results in significant package-count saving. The parallel multiplier approach was based on an expandable 8×8 CMOS/SOS array. Alternate parallel-multiplier approaches were considered too expensive or not a good match with the rest of the element. Although there are faster multipliers, utilization of their speed to achieve substantially higher element throughput would require more elaborate control logic and RAM's. For highest speed, a 24×24 bit parallel array multiplier, capable of pipeline (alternately referred to as reclocked or staged) operation can be used. The developmental 8×8 CMOS/SOS multiplier with a 100 nanosecond response time can optimally form the building block of such a multiplier as well as alternate approaches.[13,14]

A 2-stage pipelined multiplier appears optimum for that unit, with intermediate storage provided by available CMOS registers. The effective utilization of such a multiplier depends on a considerable number of gates for bus switching for its access from the RAM's and for scaling for floating-point justification. The serial/parallel multiplier approach, basically a 1×24 accumulator allows for the realization of elements over a relatively large range of performance factors by means of multiplier duplication.

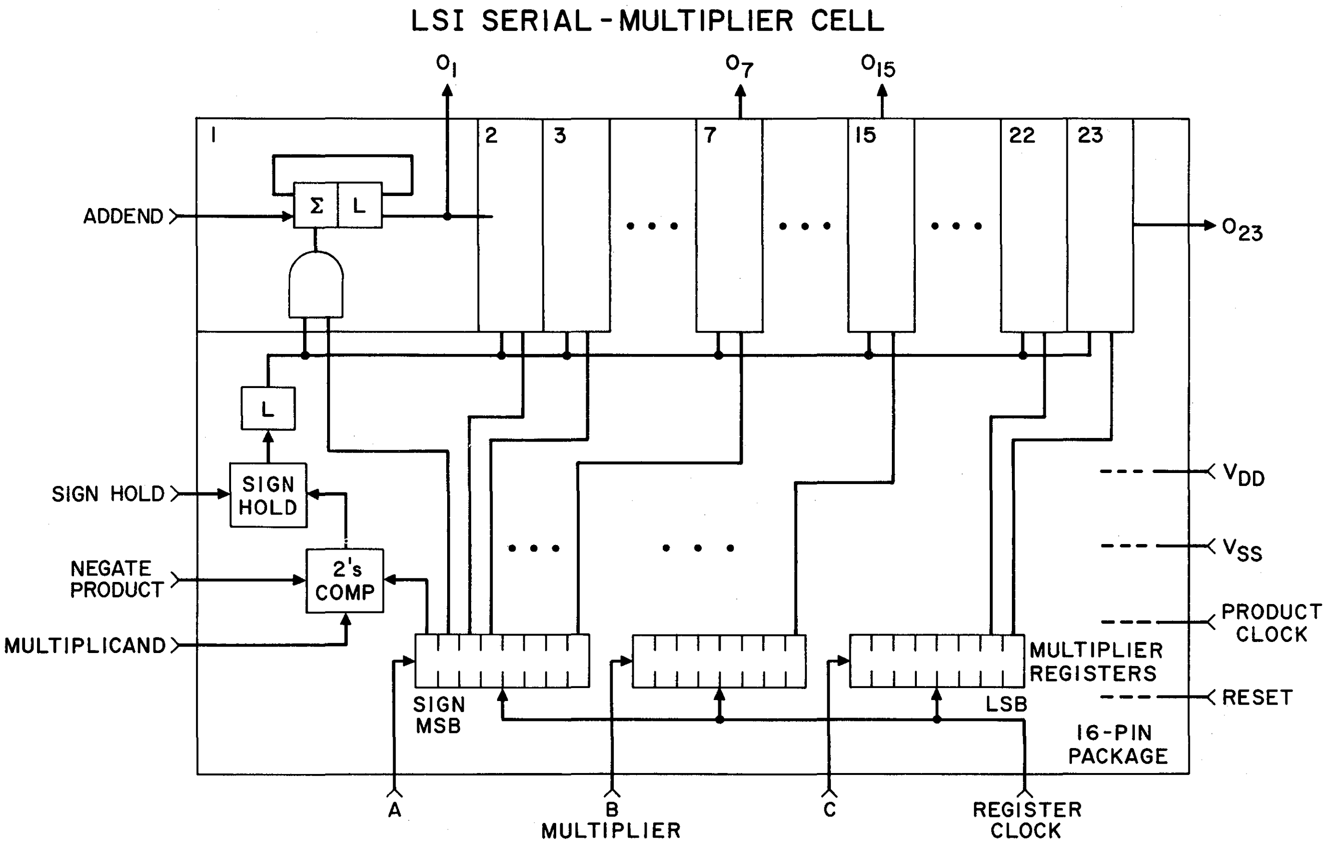


Figure 5—LSI serial-multiplier cell

On the other hand, the parallel-array multiplier locks the design into relatively high speed and cost.

This serial/parallel multiplier approach will be described, and performance factors will be summarized and compared to the parallel multiplier.

LSI serial multiplier cell

The LSI serial multiplier cell that has been developed is shown in Figure 5. The cell shown calculates terms of the form $ax+b$, where a is the multiplicand, x is the multiplier, and b is the addend. The cell shown can handle addends and multiplicands in two's-complemented form of any desired length, and a multiplier in sign-magnitude form containing a sign bit and up to 23 significant bits. Cells may be cascaded to form higher-order terms or to handle larger multipliers.

Multiplication is accomplished in the 23 adder/latch stages by successively adding the contents of the multiplicand. The outputs of stages 7, 15, and 23 are brought out so that the cell may be efficiently used with 8-, 16-, or 24-bit multipliers (the first bit is the sign). The output of stage 1 is used to gain immediate access to the product sign bit to facilitate complementing operations on the product.

The multiplier register is divided into three serial-in/parallel-out registers to provide a good compromise between multiplier input leads and the time required to load the registers.

The developmental cell contains the equivalent of 450 2-input logic gates, and has been fabricated on a chip approximately 170 mils square by means of a double-epitaxial silicon-on-sapphire CMOS process. The maximum clock rate is about 20 MHz. The cell utilizes single-phase clocks, a single power supply, is static in operation, and may be mounted in a 16-pin package.

Fixed point element

Figure 6 shows ten multiplier cells arranged as a processor to implement the general second-order multinomial (P3). Cell 1 functions as a 23-stage delay register, cells 2 through 6 function as multipliers, cells 7 through 9 function as adder/multipliers, and cell 10 functions as an adder/register. All six terms of the multinomial are generated in parallel, and delays are matched so that a single clock is used for the entire processor. The ones complementer is used to convert the processor output to sign-magnitude form before storage in the X-Y variable RAM. Only the 23 most significant bits of the output are stored. (Truncation of the least significant bits and use of a ones complementer for the conversion to sign-magnitude form will result in plus or minus one LSB error in the stored output.)

To implement general-purpose arrays, gating must be added to route data from element to element. The element control logic can then control the intercell data flow to form any given polynomial in the repertoire.

Floating-point element

A floating-point arithmetic element can also be implemented. Floating-point arithmetic requires handling of an exponent, scaling of mantissas so that bits of equal significance are added together, and left-justification of the output mantissa (with corresponding correction of the output exponent) to retain the desired floating-point format.

TRAINING AND ADAPTATION OF MULTINOMIAL NETWORKS

We now consider the methods for training and adaptation of multinomial networks that are constructed from the elements described above.[12,15] This is done with special search algorithms, generally off-line, to specify network configurations. Applications of multinomial networks generally involve operations with sensor data that are obtained as a result of "observing" a physical object, process, or phenomenon. The classical approach to design of computer models for inferences and predictions from observations has been to determine all the relevant characteristics, deterministic and/or statistical, of the process being observed, and to use these measurements (and assumptions) in design calculations. Very often the structure of the model is presumed and the design takes the form of calculating the values of certain parameters. Even if the nature of the observed process changes, the structure of the model often does not change, but the parameter values are adjusted in response to measured changes in the inputs or in the outputs.

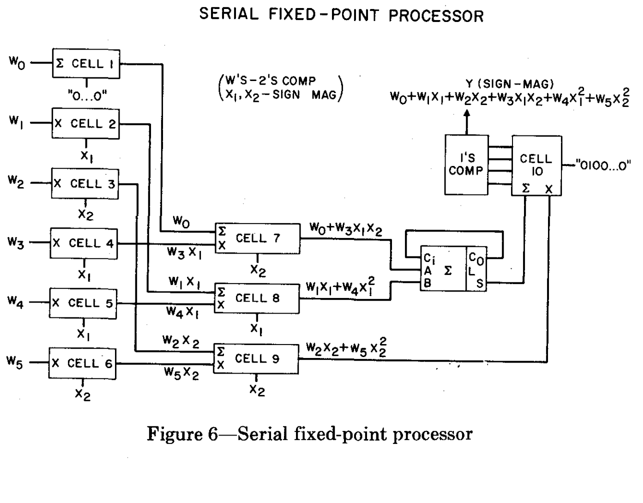


Figure 6—Serial fixed-point processor

In many important applications, the inputs (i.e., the observables) are difficult to describe analytically. The best or even a good structure for the model cannot be determined *a priori*. In this case, it is desirable to have a model structure that can adjust to representative inputs. That is, the model is *trainable* both in its structure and in its parameter values.

It is therefore desired to implement a general (usually nonlinear) function of certain input variables which we can call observables. Since little may be known about the characteristics of the observables, the parameters of the network are not known *a priori*. The network will have to be trained with representative inputs. The questions are now:

1. How should the element parameters be adjusted?
2. How should the elements be interconnected and what should their complexity (i.e., number) be?

To make the ideas clear, suppose that the input consists of N observables, $x_1, x_2, \ldots, x_N$. Also suppose that the output is a scalar whose value may be considered as the estimate of some property of the input process. In general, y will be some nonlinear function of the x_i's as follows:

$$y=f\,(x_1, x_2, \ldots, x_N)$$

Polynomial (multinomial) approximation

Under fairly general conditions, a function of N variables may be expressed in an N-dimensional Maclaurin series as follows:

$$y = a_0 + \sum_{i=1} a_i x_i + \sum_{i=1} \sum_{j=1} a_{ij}\, x_i x_j + \sum_{i=1} \sum_{j=1} \sum_{k=1} a_{ijk}\, x_i x_j x_k + \ldots \quad (2)$$

In the most general case, the coefficients, a_0, $a_i, \ldots,$ are functions of time, but for many cases of interest, the underlying characteristics of the x's do not depend on time and consequently the coefficients are constants.

Two questions which arise in the use of Equations (1) and (2) are:

1. What should the observables or measurements x_i be?
2. How many terms in Equation (2) will provide an acceptable approximation to the desired function, even though this (true) function is not known?

The answer to the first question is: All those that are *thought* to have a bearing on the desired output are used initially, and the ones which trial shows to be of little use are discarded. The second question is answered adaptively by using a trainable nonlinear network whose complexity determines the number of terms in Equation (2). The trainable network consists of interconnected elements, each of which implements a simple nonlinear function of

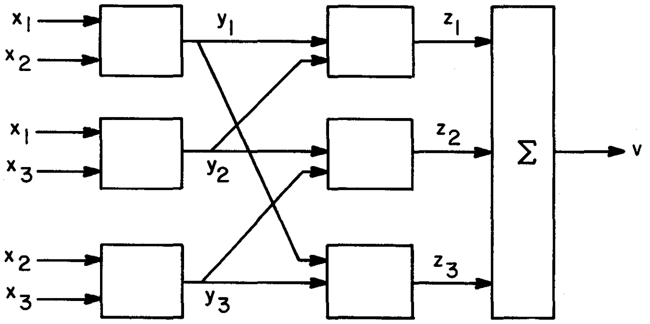


Figure 7—Two-layered network of elements

two inputs. The total network can be trained to provide an acceptable approximation to Equation (2).

A basic element of the learning network is the two-input, single-output device, illustrated in Figure 1 (and previously described) which implements the following function ($P3$) of its input x_1, x_2:

$$y=w_0+w_1x_1+w_2x_2+w_3x_1x_2+w_4x_1^2+w_5x_2^2 \qquad (3)$$

Networks of the basic element

Suppose now we consider two layers of such elements as illustrated in Figure 7. It can be seen that each z_i contains pair-wise products up to degree four. Note that the first layer contains all possible pairs of three inputs x_1, x_2, x_3. To implement a general multinomial (i.e., a polynomial in many variables) expression, the number of elements in each layer would have to grow as one proceeds deeper into the network. However, it is found empirically that acceptable approximations are obtained without this growth; in fact, the number of elements in successive layers will soon (after, say, two or three layers) decrease, until only a few are left as inputs to the final network component (which is an adder).

The known data set

Now we turn to the matter of determining the coefficients of each network element and the number and interconnections of the elements.

These tasks are accomplished with a "known" data base; that is, a data base for which the values of the dependent variable are known. The steps involved are:

1. Optimizing the coefficients in each element of the first layer;
2. Selection of those elements whose output is acceptable (rejection of poor performers);
3. Repetition of the process for each layer; and
4. A global optimization (adaptation) of all coefficients in all layers based upon the final output.

The known data base is divided into three independent but statistically similar subsets:

1. Fitting subset
2. Selection subset
3. Evaluation subset

The fitting subset is used to determine the coefficients of the elements. The selection subset is used to reject the poor performers. The fitting and selection subsets are also used for the global optimization. The evaluation subset is used to estimate the overall performance. Since the evaluation subset was *not* used for network synthesis, the performance on this subset is an accurate estimate of the ability of the network to generalize to new, previously unseen data.

Training the network

The element coefficient determinations are based, in part, upon a least-square fit to a desired output. Other criteria are of course possible and are often used. Employing a least-squares criterion, the elements are first adjusted by a matrix algebraic procedure and then by a recursive search (i.e., optimization) procedure. An outline of the steps follows.

The fitting and selection subsets are used alternately in training each layer. The fitting subset is used first to establish the coefficients. The specific observables to be used initially have already been chosen. Let these be designated by x_1, x_2, x_3, . . . , x_N. These are arranged in pairs, x_1, x_j; i, $j=1,\ldots,N$. There are $N(N\text{-}1)/2$ such pairs. Thus, the first trial will require $N(N\text{-}1)/2$ trainable elements (such as that shown in Figure 1). A pair of observables is sent to each element. The coefficients of the element are determined using a recursive search procedure with a least-squares criterion. The procedure is repeated for each of the $N(N\text{-}1)/2$ elements.

Not all of the pairwise combinations are of significant aid in extracting the desired information. The selection process, is inserted into the first layer, resulting in $R(R\text{-}1)/2$ pairs of inputs into the $R(R\text{-}1)/2$ initial elements of the second layer. The coefficients of each element in the second layer are determined as in the first layer. Then the selection subset is applied to the second layer. This will eliminate the unacceptable pairs from the second layer inputs.

The process is repeated with succeeding layers until the error rate on the selection subset is minimized. Although further reductions in the error rate on the fitting subset are realizable by incorporating more layers, to do so would produce over-fitting of the fitting data. Eventually, a single output results from each of several disjoint subnetworks. These outputs are added to produce a single output.

An example of the result of the training process up to this point is shown in Figure 8. In this hypothetical example, it is implied that at least 30 candidate parameters were initially inserted into the first layer. Only a few survived, as indicated. The figure shows that pair

(x_1, x_{28}) interacts with pair (x_4, x_{30}). But pairs (x_5, x_8) and (x_{18}, x_{29}) do not interact with each other or with the other pairs. Thus, there are three disjoint subnetworks whose outputs are added to produce a single output.

There is a final step in the training process. This is a process of vernier adjustment, or "fine tuning" of the coefficients. If the need for this vernier adjustment arises, it is because the coefficients of each element have been adjusted in the absence of interactions with other elements following them in the network. The optimum coefficient values may be different when these interactions occur. The fitting and selection subsets are also used for this final adjustment process. The vernier adjustment is a global search and may use a random search technique to obtain the final values of the coefficients. The same global search adjustment may be used for subsequent adaptation of the network.

After the final adjustment of coefficients, the evaluation subset is used to estimate the performance of the entire network.

Avoidance of overfitting is a key aspect of the training of learning networks. Good functional approximations to the fitting data subsets are obtained that are also good approximations to the data in the separate selection subsets. This means that the networks are taught to generalize properly on their experience in fitting the points in the first subsets and that error rates in later uses will therefore be small. Without avoidance of overfitting, the networks would give deceptively small errors in approximating their first sets of data and then, in most cases, do poorly on subsequent new data. We have all seen or heard of empirical models that appeared to have much promise initially, but that produced unacceptable errors when presented with new data; in most cases, such behavior is the result of overfitting. By using three independent subsets of the available data, taking care that each is statistically representative of the whole data base, the problem of overfitting is virtually eliminated and good advance estimates of operational error levels of the models are obtained.

A corollary to the guarantee that models realized by learning networks are not overfitted is the fact that these models are smoothly-fitted functional approximations. From the mathematical standpoint, they are continuous and differentiable functions, and the derivatives of these functions are close approximations to the quantitative derivative behavior of the real processes that are modeled. For this reason, one may compute numerical partial derivatives of the form $\partial y/\partial x_i$. These derivatives reveal the quantitative sensitivity of the modeled variable (y)—and thus of the process that has been modeled—to small variations about specified values of the network input variables. Once a process is modeled, or a curve fit, by computer algorithm, the hardware elements (array) described previously can be programmed for high-speed real-time evaluation of new data.

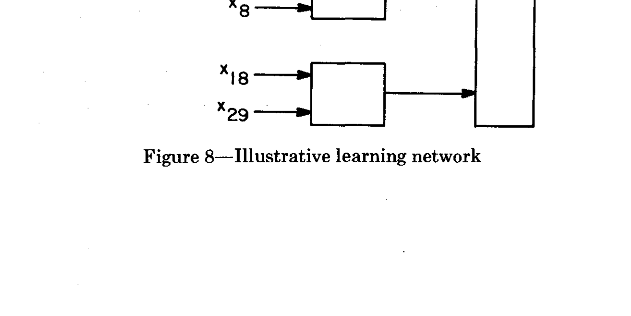


Figure 8—Illustrative learning network

AEROSPACE APPLICATIONS

Aerospace applications of the array/network concepts presented above include:

1. nondestructive inspection of structural parts
2. trajectory predictions
3. target signature classifications
4. radar refractive index corrections
5. detection of remote nuclear events
6. voice data processing
7. reconnaissance image processing
8. electronic warfare
9. avionics information systems

The status of work in several of these areas will now be summarized briefly.

Nondestructive inspection

One of the most representative applications to aerospace systems of the array/network concepts presented in this paper is that of nondestructive inspection of critical structural parts of aircrafts or missiles. The classical problem in inspection by such means as ultrasonic testing is that small defects such as fatigue-induced microcracks may be masked by reflecting surfaces such as fasteners and component surfaces. Although much theoretical work has been done to attempt to characterize the pulse échoes obtained from microcracks and other defects, no all embracing theoretical formulation exists at this time. The adaptive training of a multinomial network offers an attractive approach to the processing of pulse echoes in the complex signal environment typical of ultrasonic inspection applications. Data may first be gathered on test specimens having known properties and used to train the network in accordance with the procedures outlined above. Such work is progressing at the present time. It has already been demonstrated that the adaptive learning network can be trained to classify correctly flat-bottom hole defects in 7075-T6 Aluminum test blocks, using transducers of various diameters and differing band-pass characteristics, centered in the neighborhood of five megahertz. As reported in Reference 16, in a total sample of 48 flat-bottom hole defects, 46 were correctly classified by this technique. An important aspect of the use of the multinomial network method is that it is not necessary to specify *a priori* the classifier input parameters that are most informative for a given application. In the case of the ultrasonic nondestructive inspection application, 96 candidate waveform parameters were considered during network training. Fifteen of these parameters were found to be relevant to the flat-bottom hole classification. These parameters are those that describe the overall shape and content (area) of certain parts of two waveforms computed from the pulse echo waveforms—the Power Spectrum and its log Fourier transform, the Cepstrum. Interestingly, maximum amplitude of the time waveform was not found to be a discriminating parameter when the transducer and/or transmission medium were subject to variation. The resultant network structure found for flat-bottom hole defect classification consists of 13 elements containing a total of 78 coefficients, and implementing an eighth degree function of the 15 input variables.

If application to the fatigue crack specimens is also successful, the adaptive learning network will provide an inspection tool of great value to the maintenance of aircraft and missiles in the operational inventory. The work to date has already demonstrated that the new procedures allow one to synthesize signal processors that deal with waveforms from the real environment and that learn which parameters of these waveforms are the most relevant, while combining these parameters into an adaptively trained signal classifier. Other applications have been suggested for the methodology in the related areas of acoustic emission testing, computer-aided manufacturing, optimization of metal removing processes, control of forging and casting, inference of material physical properties from microstructure data, and forecasting of maintenance necessitated by stress corrosion deterioration of flight vehicle structures. In each instance, the key point is that networks can learn to infer or predict from the natural data that are produced by the processes. These "natural" data may be records of process sounds, vibrations, deterioration due to corrosion, etc.—anything readily accessible for economical instrumentation or recordkeeping.

Trajectory predictions

Successful R&D has been conducted for more than a decade for application of multinomial networks to the prediction of aerospace vehicle trajectories. In summary, these investigations have established that the network methods are capable of inferring vehicle parameters, such as ballistic coefficients, quite accurately. Additionally, the networks make extremely fast, accurate predictions of object trajectories. These predictions are comparable in accuracies to the conventional procedures whereby equations of motion are integrated in serial computers, but are very much faster because of the computing speed of the parallel network structure.

Target signature classifications

Attention is being increasingly directed toward the automatic classification of target signatures from single or multiple sensors. An example of this work is that of classifying the sources of ground vibrations and/or acoustic emissions monitored by sensors in air drop ordnance. Very promising results are being obtained in such work.

Radar refractive index corrections

Reference 17 presents a new approach to the problem of computing height correction for aircraft or other objects tracked by surface radars. The basic procedure, when using the multinomial network, is to operate a cooperative aircraft that is equipped with an accurate radar altimeter in those regions of airspace for which the true altitudes of unknown targets are to be obtained. The cooperative target is tracked and an adaptive learning network is used to model the relationship between observed range, elevation angle and azimuth angle, and the independently measured height of the cooperative target. It is then possible to interrogate the network very quickly whenever the true height of an unknown target is to be computed. The inputs to the network become the apparent range elevation and azimuth of the unknown target, and the network output is the estimated true height of that target. The structure and coefficients of the network are adjusted adaptively as atmospheric conditions change. The reference presents a comparison between the accuracy of this approach and that of the conventional procedure. The height error, using the new approach, is approximately ⅓ of the error obtained with the prior state-of-the-art method.

Detection of remote nuclear events

Reference 18 presents results of work performed to assess the accuracy of an adaptive learning network classifier for discrimination between remote underground nuclear events and deep-core earthquakes (which masquerade as nuclear events in many cases). The results show that nearly perfect discrimination between the two classes of remote events is obtained.

Voice data processing

The application of adaptive learning networks to the identification of spoken languages is discussed in Reference 19, which presents the results of an investigation into multinominal networks used to generate nonlinear features of 29 phoneme and phoneme-like parameters obtained from speech waveforms. The networks are trained to discriminate between each pair of languages in the set of languages to be identified. The outputs of the networks are then input to a decision logic to identify which language is being spoken. In the case of five languages to be discriminated, this means that a group of nonlinear transformations is produced by 10 individual networks, each of which maps the 29-dimensional input space onto a component of a 10-dimensional output space. The structure of a typical trained network is shown in Figure 9. The results of the cited investigation show that significant improvement is obtained in the accuracy of language identification and in insensitivity to idiosyncracies of individual speakers.

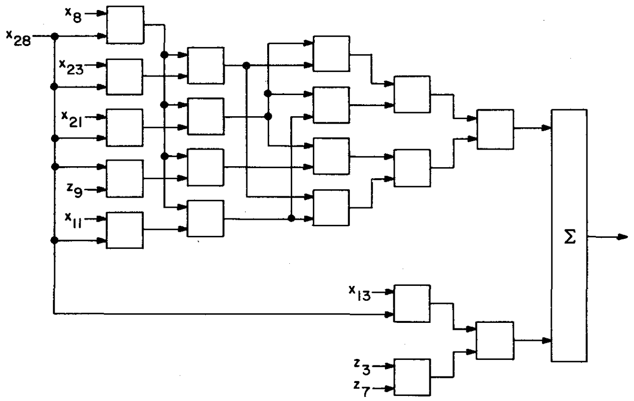


Figure 9—Multinominal network classifier that discriminates between two languages. The x's and z's are phoneme and phoneme-like parameters from the special waveforms

SUMMARY AND CONCLUSIONS

Elements for use in programmable arrays have been defined, and design approaches and performance indicated for alternate organizations. By giving an element the capability of evaluating one of five multinomial expressions with pre-set coefficients, upon command, and by providing means for interconnecting arrays of elements in various configurations, a number of signal processing functions can be realized. These include:

Classification, Prediction and Control
Hypersurface Computation
Digital Filtering (Recursive, Transversal)
Transformations

The arrays can, in fact, be alternately and rapidly reconfigured to do any of the above types of tasks. In that the transformation coefficients and multinomial degree and form can be readily controlled, an array can be trained to suit a particular situation.

Two types of design approaches were studied for realizing the hardware of a programmable element, one based on a parallel-pipeline multiplier and the other on a serial-type multiplier. In each case the multipliers would be realized by appropriate custom LSI, CMOS/SOS multipliers.

Considering that some or all of the speed differences between the two approaches can be compensated for, when required, by using a greater degree of parallelism in a net organization (Figure 2), the serial-multiplier approach was recommended for development. With a full 32-bit floating point (24-bit mantissa, 8-bit exponent) capability total of 10 of the custom LSI CMOS/SOS serial-type multiplier packages supported by 15 packages accommodating custom LSI control chips, a full six-term multinomial of 2-inputs can be evaluated in 5.2 microseconds. For 16-bit fixed point capability the control chips reduce to two, and the computation period reduces to 2.9 microseconds. The random-access memory to support each element will depend on its degree of multiplexing. For a fully populated array, up to 3 RAM packages would be required per element.

The advantages of such arrays include:

1. High-speed capability for general-purpose digital processing. The speed can be tailored by providing varying degrees of parallelism of the element in an array.
2. Capability for relieving software problems. The array can be considered as firmware once a set of coefficient and interconnect vectors have been found, or are known, for a particular application.
3. Capability for providing fault-tolerant computing. In general, alternate coefficient and interconnect vectors exist to realize the same transformation. Hence, if an element in a fully populated array is faulty, reprogramming of the array is possible by by-passing any faulty element(s). The degree to which this bypassing is done will, of course, depend upon the number of elements available and the complexity of the transformation.

ACKNOWLEDGMENTS

The guidance provided for this program by Mr. C. W. Gwinn of the U.S. Air Force Avionics Lab. is appreciated. Portions of this work have been performed for the Air Force Systems Command, United States Air Force, under Contract F33615-73-C-1089 to RCA with Adaptronics, Inc. subcontractor. The other portions of this work have been performed by Adaptronics, Inc. under the sponsorship of Armco Steel Corporation, RCA, the U.S. Air Force, and other organizations. Assistance of H. Urkowitz and D. A. Miller of RCA and A. N. Mucciardi of Adaptronics, Inc. in aspects of this work is acknowledged.

REFERENCES

1. Gilstrap, L. O., Jr., H. J. Cook and C. W. Amstrong, *Study of Large Neuromime Networks,* Adaptronics, Inc. Fin. Rept. Contract

AF33(615)-5125, AF Avionics Lab., AFSC, AFAL-TR-67-316, AD#824 470, Dec. 1967.

2. Gilstrap, L. O., "Keys to Developing Machines with High-Level Artificial Intelligence," presented to *ASME Design Conference,* New York, April 22, 1971, (ASME paper #71-DE-21).
3. Cleveland, D., "Hardware Realization of Trainable Multivariable Nonlinear Transformations," presented to *Computer Designers Conference,* Anaheim, Calif. January 19-21, 1971.
4. Mucciardi, A. N., "Neuromime Nets as the Basis for the Predictive Component of Robot Brains," presented to *ASC Fourth Annual International Symposium,* Washington, D.C., October 8-9, 1970; *Cybernetics and the Management of Ecological Systems,* Robinson (Ed.), Spartan Books.
5. Ivakhnenko, A. G. et al., "Group Handling of Data in Identification of the Static Characteristics of a Multi-Extremal Plant," *Soviet Automatic Control,* 14, 2, 1969, pp. 30-37.
6. Ivakhnenko, A. G. and Y. V. Koppa, "Stochastic Algorithms and the Group Method of Data Handling in Prediction of Random Events," *Soviet Automatic Control,* 14, 3, 1969, pp. 20-32.
7. Ivakhnenko, A. G. and Y. V. Koppa, "Algorithm of the Group Method of Data Handling Using Linear Operators," *Soviet Automatic Control,* 14, 4, 1969, pp. 50-58.
8. Ivakhnenko, A. G., "Polynomial Theory of Complex Systems," SMC-1, 4, October 1971, pp. 364-378. *IEEE Trans. on Systems, Man and Cybernetics.*
9. Hampel, D., R. W. Blasco and D. Cleveland, "Electronically Programmable LSI Arrays," *NAECON '74 Record,* pp. 134-141.
10. Gilstrap, L. O., Jr., "An Adaptive Research to Smoothing Filtering and Prediction," *Proc. of 1969 NAECON,* pp. 275-280.
11. Barron, R. L. and C. W. Gwinn, "Applications of Self-Organizing and Learning Control to Aeronautical and Industrial Systems," presented to *ASME Design Engineering Conference,* New York, April 19-22, 1971. (ASME paper #71-DE-22).
12. Barron, R. L., "Theory and Application of Cybernetic System: An Overview," *NAECON '74 Record,* pp. 107-118.
13. Hampel, D., L. Micheel, K. Prost, "Threshold Logic Multiplier," *NAECON 1973 Record,* pp. 288-295.
14. McIver, G. W., R. W. Miller, T. G. O'Shaughnessy, "A Monolithic 16×16 Digital Multiplier," *ISSCC Digest,* Feb. 1974, pp. 54, 55.
15. Barron, R. L., "Applications of Learning Networks in Computer Aided Prediction and Control," *SME CAD/CAM Conference,* February 12, 1975.
16. Mucciardi, A. N., R. Shankar, J. Cleveland, W. Lawrie and H. L. Reeves, *Adaptive Nonlinear Signal Processing for Characterization of Ultrasonic NDE Waveforms,* Interim Report, Task 1, Inference of Flat-Bottom Hole Size, Adaptronics, Inc. January, 1975.
17. Barron, R. L., F. W. van Straten, and R. F. Synder, "Inference of Refractivity Structure and Generation of Ray Traces by Analytical and Adaptive Methods," *1975 NAECON,* To be Presented.
18. Mucciardi, A. N., *Demonstration of Capability of Adaptive Learning Networks to Discriminate Automatically Between Earthquakes and Nuclear Explosions Based on Seismic Parameters,* Adaptronics, Inc., ATN-76, April 23,1973.
19. Mucciardi, A. N., E. C. Orr and R. Shankar, *Discrimination between Five Spoken Languages by Trainable Hypercomp^{TM} Network Classifier,* Adaptronics, Inc. Final Report for RCA Corp., April 22, 1974.

Software reliability—A method that works

by R. H. THAYER
Space and Missile Test Center
Vandenberg AFB, California

and

E. S. HINTON
Logicon, Inc.
San Pedro, California

INTRODUCTION

Software reliability is receiving increased attention from a broad spectrum of computer users as larger computer programs continue to be implemented in diverse and widespread areas. The reason is fundamental: software reliability has been poor on many large systems and poor on systems which have a high degree of human interaction.

The Department of Defense has had several systems with significant software problems. Current Air Force systems have suffered development and maintenance difficulties that were directly attributable to poor software reliability. The Strategic Air Command's command and control system (465L)[1] and the F-111 Mark II avionics system[2] are both examples of such systems. Many missile systems had software error problems that, at one time, gave program managers cause for concern. It is not surprising, therefore, that many attempts have been made by the Air force to improve the software reliability picture, especially in the space and avionics system areas. Several management and technical approaches, including those that have worked well for hardware, are being used and developed with varying degrees of success.[3]

One of the most successful approaches to improving software reliability that the Air Force has used to date, particularly on highly critical real-time software, is the utilization of an independent software test agency, to perform analysis, testing, and overall evaluation on the system.[4,5]

In fact, the improvement in reliability as a result of independent analysis and testing of software has prompted the Air Force to include the following policy statement in its new regulation, AFR 800-14, dated 10 May 1974:

> *"Program Management Directives require, and Program Management Plans provide for: Establishment of computer technical and managerial expertise responsive to the Program Office which is independent of the system prime or computer program development contractor and, preferably, an organic capability of the Program Office."*

Independence is the key ingredient in this analysis and testing function, and in the context of the discussion, implies that the function is accomplished by an agency whose sole purpose is the detection of errors in the developed software, i.e., the agency has no development responsibility.

BACKGROUND

The beginning

The independent analysis and testing concept has been evolutionary in development, and it will continue to evolve as new and varied analysis and test techniques and tools are designed. Initial conception of the "independent" approach is believed to have been the result of critical software failures on early missile and space launches. The first application of the approach is believed to have been on the software associated with the Titan II and III during the 1962-1964 time period.

In the critical area of missile flight safety, instances have been reported where range safety system software has failed causing costly delays of launch operations. Furthermore, software problems have been detected which could have produced a situation where a good launch would appear bad to the missile flight control officer. Such situations would require the destruction of an expensive vehicle and payload.[6]

The costs associated with such failures prompted the use of, what in 1971 were then considered to be, drastic actions. The Air Force, considering the expense of failed launch missions, proceeded to hire an independent contractor to analyze the software in a "validation" process, prior to releasing the software for operational use. It was felt that an outside or fresh look at the software should reveal faults, whereas the development contractors and programmers were "too close to the problem" to come up with an objective analysis. This approach proved to be highly successful and this method of software validation has been used at Vandenberg AFB from 1971 until the present.

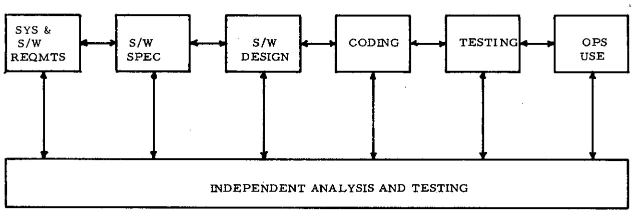


Figure 1—Software development process with independent analysis and testing

Present usage

The confidence gained from successful missile launches and other developments that utilized the dual contractor approach prompted continuing it as a standard procedure on critical, high-risk software packages. Examples of DoD systems which use and have used an independent agency for software testing and analysis are Safeguard, Minuteman II and III, Titan, and B-1.

In general, most of the independent analyses and testing performed to date have been done on real-time control programs. There are however, extremely critical programs, such as the real-time software that controls nuclear weapon systems, which have become prime candidates for testing and evaluation by an independent agency. This procedure is now standard Air Force practice on such systems and is prescribed in Air Force Regulation 122-9, dated 19 July 1974. The fact is, that on the most critical software with which the nation is involved, that which controls nuclear weapon systems, analysis and evaluation by an agency independent from the software developer is mandatory. This strongly implies that it is one of the most effective known ways to achieve software reliability.

DESCRIPTION OF THE INDEPENDENT TESTING AND ANALYSIS METHODS

A software development life cycle is illustrated in the block diagram of Figure 1 and can be considered a chain of events beginning with a customers requirements and ending with the operation and maintenance of the software through the programs useful life. Independent Testing and Analysis, (IT&A), is a nonspecific set of analyses, studies, tests, and evaluations that is conducted by an independent agency on a computer program that are intended to verify compliance with requirements and show correctness of programming. IT&A does not absolutely guarantee verification or correctness of software, but it improves confidence that the program can be depended upon to perform reliably in an operational environment.[4,5] As illustrated in Figure 1, independent analysis and testing is most effective when applied across all phases of this development. From the developer's point of view, software development is a serial process with frequent iterative loops between adjacent phases. The independent tester, on the other hand, is not in the direct chain of development, and he analyzes the output of each phase as to its ability to satisfy input requirements, e.g., specification to design, or design to coding. He performs this function for all phases as soon as the output becomes available, again with the sole purpose of detecting errors.

INDEPENDENT TEST AND ANALYSIS WORKS

Some proof

Independent Test and Analysis can find errors that are not found through ordinary software testing.

A software reliability study[7] recently completed for the Advanced Ballistic Missile Defense Agency has pointed out the importance of early error detection. In the study report, software errors found through independent analysis on eleven separate projects were categorized as being catastrophic, serious, moderate or trivial. These categories were defined essentially as follows: catastrophic errors which would terminate the program execution; serious errors which could severely degrade the program performance but would not be fatal; moderate errors which would not have major impact on program performance; and trivial errors which would have no effect on program performance, e.g., flow chart did not match code, though code was correct. A summary of the errors discovered, by severity, is shown in Table I.

Significantly, many errors which might initially have been considered trivial, if not discovered early in program development, would have created error conditions of much greater severity. Independent analysis discovered errors in released specifications that, at that time, were considered trivial because of the relative ease of correction, hence the 553 trivial errors in the top row of Table I. The effect of implementing those "trivial" errors into program design and code would have been to increase catastrophic, serious and moderate errors to the numbers shown in the bottom

TABLE I—Effect of Implementation of Specification Errors on Eleven Software Projects

	ERROR SEVERITY				
	CATASTROPHIC	SERIOUS	MODERATE	TRIVIAL	TOTAL ERRORS
ERRORS DISCOVERED EARLY IN SPECIFICATION	21	149	479	553	1202
ERRORS ANTICIPATED AFTER IMPLEMENTATION	36	200	558	408	1202

row. Trivial errors would have decreased to 408 as they were reflected in more serious categories.

These trivial errors were concerned primarily with reqirements and specifications. In fact, 31 percent of all non-trivial errors would have been attributable to errors in specifications. The significance of the independent analysis actions is pointed out by the fact that the specifications had been reviewed by the developer and had been released, and discovery of the errors came as a result of the independent review.

Obviously, if the software specification contains an error with respect to the system software requirements, the correction will cost much less before design, coding and testing are accomplished than after. For this reason, early establishment of the independent agency is likely to provide greater cost savings.

Some more proof

One data point that may provide some insight to quantitative benefits of IT&A concerns the results of devlopment, independent testing, and operational use of a highly critical, real-time control system.[8,9] Several hundred errors were found by the developer in two versions of three separate programs and corrected during development and initial testing. Such a situation could be considered normal on any large software development.

However, sixty additional errors were found during IT&A, several of which would have been catastrophic. Since release to the operational agency, one error has been detected in one year's use. This error was non-critical. As another single example, this instance would tend to support the effectiveness of the use of an independent agency for testing critical software.

And more proof

A 25,000 word program, which was integral to the range safety system for Vandenberg AFB missile launches in the 1960's, was completed and turned over to the Air Force. This program was operational for approximately eight years with many modifications being made in response to changing requirements. At that time, an independent agency was charged to perform detailed IT&A of the program and 20 errors were detected, seven of which were critical.[6] In another case a new range safety system was delivered to Vandenberg AFB. This system, after undergoing IT&A, was determined to be unready for operational use and subsequently had to undergo considerable modification.[10]

More proof needed

Although Independent Test and Analysis has proven to be an excellent method of finding software errors, there is little data available on the completeness of this approach. There are several reasons for the lack of data: one being the added cost for performance of testing beyond that accomplished by the independent test group. Since such testing and resultant data are not available, a comparison of the software reliability (error-freeness) before and after independent testing is not quantitatively possible in this paper.

Another reason comparative data is not available is that IT&A has only been accomplished on highly critical programs. Therefore, emphasis has been on performance of testing and achievement of reliability on a program by program basis, not on the gathering of data for analysis of how well the testing was done.

Research into the overall problem of software reliability has begun to examine the various parameters associated with software development and testing which seem likely to affect reliability. One such study effort is being undertaken by the Rome Air Development Center for the Air Force.[3] It is hoped that the results of this and other analyses will provide enough data so that quantitative assessment of the software reliability associated with the various testing approaches will be possible.

INDEPENDENCE TEST AND ANALYSIS TECHNIQUES

Analysis and testing techniques have also been evolutionary in their development. Early independent analysis consisted largely of "manual" code review, comparison of logic flows with listings and execution of modules whenever possible. These same techniques are still basic to the overall process, though many tools have been developed which automate many of its aspects. Examples of these tools are automatic flow charters, compare programs, logic/equation generators, structure analyzers and timing analyzers. Table II lists several of these with their associated characteristics.

Some of the more interesting work being accomplished in tool design and development is in the area of automatic verification systems, including test sequence generators, code segmenters, and path execution counters which gather statistics concerning program structure and path execution. Automated tools are now being developed and tested which analyze program structure and produce normalized flow diagrams which represent the actual program as if it were coded in accordance with the rules of structured programming. In addition, the test program then produces logic and arithmetic statements which represent the effect of the code execution.

Analysis and testing processes have become more sophisticated as experience has been gained and new tools designed. This is reflected in the increased use of simulation as a testing tool. Modern test configurations may use a closed-loop system involving six-degree-of-freedom vehicle models, environmental simulations, simulated system interfaces and, when actual flight computers are not available, interpretive computer simulations which run on general purpose host computers. These simulations may be modified forms of those used by the software

TABLE II—Software Testing Tools with Associated Characteristics

TOOL	PHASE USED	INPUT	OUTPUT
Flowcharter	Pre Execution	Source Code	Logic flow diagrams
Comparator	Pre Execution	Two versions of Source code	Locations of all differences
Structure Analyzer	Pre Execution	Source code and pattern	Locations of all occurrences of specified pattern
Timing Analyzer	Execution	Source code & software probes	Module execution times
Logic/equation generator	Pre Execution	Source code	Logic/Equations
Correctness proof	Pre Execution	Source code	Statement of equivalence or non/ equivalence
Syntax Analyzer	Pre Execution	Source code	Syntax Error listing
Test Sequence generators	Execution	Source code	Test sequences
Path Execution counters	Execution	Source code	Execution data
Code segmenters	Pre Execution	Source code	Code segment designations

developer or they may be designed and produced by the IT&A agency for its sole use.

WHY INDEPENDENCE

The greatest single advantage of IT&A over the more conventional approach of having the developer perform all software testing is that a new and different point of view is established with regard to the reliability of the software. Development personnel must hold to cost and schedule constraints with the fundamental goal of "making it work, within cost and on schedule." Their philosophy is, and has to be, oriented toward showing that the software is correct, i.e., it performs all intended functions and performs no unintended ones. This is not the best frame of reference from which to detect software errors. The independent agency is brought aboard explicitly to analyze and test the developed software. His sole goal is to find errors and bring them to the attention of the developer so that they may be corrected. He assumes errors exist in the software and his whole reputation for thoroughness and effectiveness rests on his being able to find them. With this point of view, the independent analysis and test agency has much stronger incentives to find all errors than the unaided developer.

A beneficial side effect contributed by use of an independent group is the establishment of a competitive atmosphere in which the developer, knowing his work is being independently checked, strives to make fewer errors, while the independent agency tries even harder to find them. The professional pride of each group contributes to the building of the competitive environment and software quality benefits as a result of the interaction.

Another advantage associated with independence in analysis and test is that the tendency for management (always success oriented) to curb test efforts in the hope that the software is "good enough" is greatly reduced. When development and testing are under common management control, pressures usually exist which dilute the effectiveness of the analysis and testing function. An independent agency with a stated goal of detecting errors in the software being developed is in a stronger position with regard to management influence.

A third advantage of IT&A results from the natural effects of specialized experience gained from "doing the job." Effective analysis of software documentation and code requires insight into requirements/specification relationships, methods of functional allocation, code analysis techniques, and simulation capabilities. These are "learned" aspects of the analysis and test process. When an independent agency is available that specializes in this type of work, learning time is reduced and effective effort begins immediately upon assumption of the task.

REDUNDANT PROGRAMMING

Another, and somewhat related, approach to reliable software is redundant programming (also called parallel programming and dual programming).[11] This concept gets its name from the redundant systems used by the hardware engineers to increase hardware reliability. Here, two separate and independent programs, which can be executed independently to perform the same function, are produced by separate development teams; both programs are run at the same time on separate hardware systems or sequentially on a single system. In case of software failure of one system, the other system would take over.

A less drastic approach is to again have two programs, functionally identical, developed by independent programming teams. Here only one program would be used at a time and this could be parallel programming rather than redundant programming. One can be a back-up in case the other fails. Comparison of outputs can detect errors in one or both. Competition in development adds incentive for the production of a good program. Two knowledgeable groups are available for further effort in the area, eliminating the "sole source" situations. Two groups are more likely to establish a reasonable delivery time for the product than one working alone, again because of competitive pressure.

This approach also has the key ingredient of independence. Completeness of testing through the use of automated tools is not implicit in this method, though it certainly could be incorporated by both programming teams. The parallel approach is the result of philosophy that programs are too complex to be tested "adequately," and that matching comparisons of results from the efforts of two separate programs designed to perform the same function is a valid and sufficient test.

Overall, the advantages of parallel programming may outweigh the disadvantages, though the real question of how much this approach improves software reliability, has not been answered. Whether more independent testing on one program would produce higher reliability than normal development testing on two programs that are functionally the same is not clear. As software becomes more expensive and reliability more critical, both approaches may be used on the same development effort.

DISTRIBUTION OF SOFTWARE DOLLARS

In time of reduced budgets and spiraling costs, the concept of dual analysis and testing, independent or not, is likely to raise the budgeteering eyebrows, and when dollar cuts are inevitable, such activity may get immediate nomination for the proverbial "ax." This is particularly true when management has not developed "software awareness." Expenditures for software have been increasing dramatically, not only in an absolute sense, but relative to the associated hardware. According to the frequently-referenced Air Force Study,[1] published in 1972, software costs were three times that of hardware and the ratio was increasing continually. Based on the labor-intensive character of software and the decreasing cost of machine processing capability, current inflationary trends will undoubtedly increase the software-to-hardware cost ratio even more rapidly than predicted.

The referenced study also reported that, as an average, the effort expended on software was divided into three phases on a percentage basis as follows: design and analysis, 40 percent; coding, 20 percent; checkout and test, 40 percent. These percentages have now become almost classic throughout the software development community. However, they refer in general to the development costs incurred by the software development contractor. The concept of analysis and test of software by an independent agency implies an additional cost of some magnitude, though it may have a wide variance with respect to the initial development cost, normally ranging from 30 percent to 60 percent. The higher costs imply considerable analysis effort and increased use of specially constructed software test tools, whereas the lower costs are associated with a less active, more monitor-oriented activity. In one avionics software program, the independent monitoring activity was initially established at a 10 percent level, and was limited mostly to management support. Later emphasis was increased on independent analysis and test, and the relative percentage was raised to approximately 30 percent. Costs associated with such analysis and test are shown for 10 software development efforts in Table III.

TABLE III—Relative Costs of Development and Independent Analysis and Test for Ten Programs

NAME	SIZE	DEVELOPMENT COST (MM)	INDEPENDENT ANALYSIS & TEST COST	PER CENT
A	LARGE	750	252	33
B	LARGE	750	300	40
C	MEDIUM	337	142	42
D	MEDIUM	310	150	48
E	MEDIUM	112	37.5	33
F	SMALL	12.5	6.25	50
G	SMALL	12.5	6.25	50
H	MODIFICATION	125	50	40
I	MODIFICATION	650	450	69
J	MODIFICATION	300	112	37

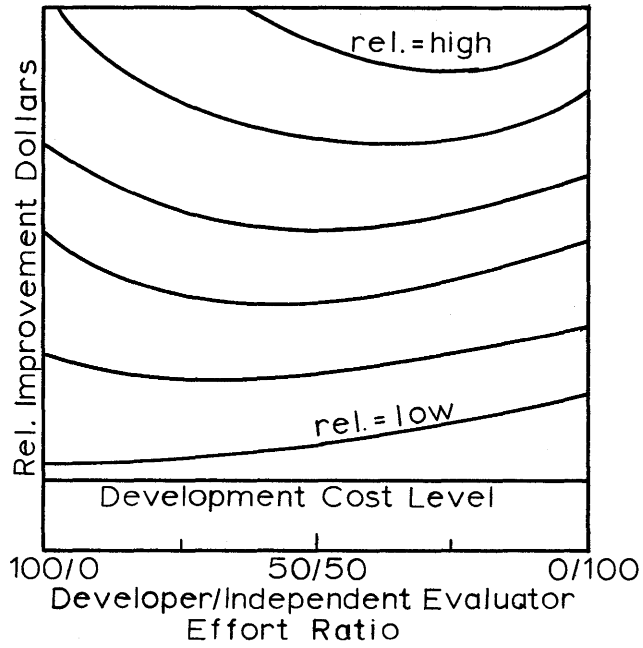


Figure 2—Software reliability as a function of improvement dollars and developer/independent evaluator effort ratio

The total dollars available for software development can be allocated in various ways. Assuming that reliability is a prime requirement in any software procurement, one should be able to add some amount to the developer's contract and achieve an incremental increase in error-freeness. An alternative approach would be for one to take that amount and apply it totally to an independent contract. All intermediate variations are, of course, other alternatives. Which alternative is the most cost-effective, depends on several factors, such as status of documentation, uniqueness of system, and availability of test tools. Generalized relationships are depicted in Figure 2 which are believed to hold. In all cases, some level of reliability and dollar costs are basic as a result of the initial contract, i.e., a specific expenditure is indicated as a horizontal constant-dollar line in Figure 2. The curves indicate a level of specific reliability or error-freeness, and they show that by splitting the dollars spent to obtain significant software reliability improvement, one is more likely to achieve higher reliability than by spending it on either the developer or the independent agency alone.

The curves are likely to be assymetrical or skewed with respect to the 50/50 percent abscissa since several factors affected the specific cost relationships for a particular project. As an example, if the developer has not produced and maintained adequate documentation during the development process, an independent agency may have a difficult task when assigned to do analysis and testing. Funds for preparation of the documentation would likely be most effectively spent with the developer. The result might be the skewing of the curve for a specific level of reliability to the left. If adequate documentation is available and initial integration testing has been completed, the curve will probably be skewed right.

CRITICAL SOFTWARE IN THE BUSINESS WORLD

Computer software made its original debut in the field of accounting, and the greatest use of software still remains within the business world of commerce and industry. Military application of computers and their associated software began shortly thereafter and has steadily increased, not only in the business-oriented payroll and accounting areas, but more significantly, in the area of real-time control of complex equipment such as missile, satellite and aircraft avionics. The software in such systems frequently affects the safety of personnel and expensive equipment, and all reasonable steps are taken to insure its reliability.

Though personnel safety is not normally involved in the operation of commercial software, the dollar costs associated with software failure can be significant. This is particularly true in view of the increased size of business systems, the increased use of remote time-sharing terminals over widely distributed geographic locations and the trend toward control of fund disbursement directly by computer. Interactive systems with distributed time-sharing terminals require complex system software and failure modes can take many forms which may not be obvious to the system user or management. As the complexity increases, the risk of failure also rises, and with fund control being handled increasingly by machine, risk of fiscal liability will undoubtedly increase.[12]

For clarification, the definition of failure may require some rethinking as it applies to business-oriented software. However, the concept that software fails if it does not perform its intended function or it performs unintended functions, should serve equally well in both the military and commercial world. With that definition in mind, software that writes checks to fictitious companies for services not rendered could be said to fail, at least as far as management is concerned.[13] It may be that the software has been deliberately designed to produce such unearned checks. In this case, though it might be highly reliable in the eyes of the programmer, management undoubtedly would have a different view. Of the various approaches tried by the military software managers to insure software reliability, the one most suited to preventing the insertion of an "intentional" error is the utilization of an independent agency to perform an in-depth analysis and test of the program. In view of the increasing number of news accounts regarding computer associated fraud or embezzlement,[13] and the difficulties that appear to be arising in the employment of computers in vote-counting,[14,15] it is anticipated that independent analysis and testing of software and associated systems will soon become commonplace in many areas in the business world.

SUMMARY

Although independent test and analyses does not guarantee absolutely error free software, it has become an important approach to achieving highly reliable software in the Air Force. It has been used successfully on many critical programs particularly those involved with missile and space systems. The independent aspect of the function is the key characteristic and it offers the advantages of a new and objective point of view, freedom from unrealistic management constraints and the opportunity for the software manager to use personnel experienced in software testing.

Commerical software and computer systems have many characteristics which would tend to make independent analysis and testing attractive to the business world. Extensive application of the principle to critical commercial systems is anticipated for the near future.

ACKNOWLEDGMENTS

The authors wish to acknowledge the many informative discussions with James C. Alexander with regard to the history of operational software at Vandenberg Air Force Base. Acknowledgment is also given Ms. Barbara Avillanoza for her excellent typing and redaction.

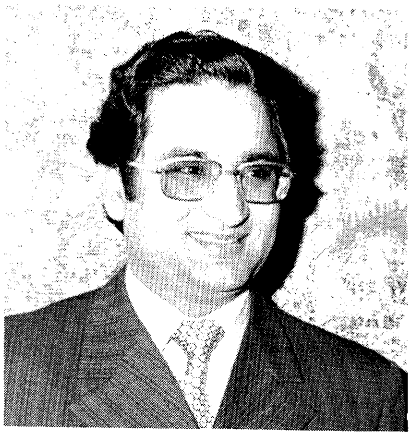
REFERENCES

1. *Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980's,* United States Air Force, 1972.
2. *F-III Flight Programs Support Study,* General Dynamics Report, FCE-1152, Jan 1972.
3. Thayer, R. H., *Rome Air Development Center R&D Program in Computer Language Control and Software Engineering Techniques,* RADC TR-74-8, April 1974.
4. Rodrick, T. L., "Verification and Validation of Aeronautical Systems Software," *Proceedings of the Aeronautical Systems Software Workshop,* April 1974.
5. Wattenberg, R. E., "Independent Test and Evaluation," *Proceedings of the Aeronautical Systems Software Workshop,* April 1974.
6. Burke, J. P., *et al., Verification and Validation of the Vandenberg Prediction System—Version 30,* Logicon Report No R:SEA-71031-VIPS-134 Oct 1971.
7. Dana, J. A. and J. D. Blizzard, *Verification and Validation for Terminal Defense Program Software,* Logicon Report No. HR-74012, May 1974.
8. Fujii, R. U., *el al., Command Data Buffer Operational Executive Program,* Final Report, USAF Space and Missile Systems Organization, Mar 1973.

9. Fujii, R. U., *et al., Performance Analysis and Technical Evaluation of the Minuteman Operational Targeting Program,* Final Report, Secret Classification, USAF Space and Missile Systems Organization, Nov 1974.
10. Bartlett, J., *et al., Verification and Validation of the Mobile Range System Impact Predictor,* Logicon Report No R;DS/72-107/1141, Sept 1972.
11. Glib, T., "Parallel Programming," *Datamation,* Vol 20, No. 10, Oct 1974.
12. Alexander, T. "Waiting for the Great Computer Rip-off," *Fortune,* Vol. 90, No. 1, July 1974.
13. Farr, W., R. T. Cooper, and J. Belcher, "Plot to Loot Los Angeles of $2.5 Million Fails, *Los Angeles Times,* Dec 8, 1974.
14. Myers, E., "Election Day Log," *Datamation,* Vol 18, pp. 90-93.
15. Farmer, J., "Computerized Voting: Many Happy Returns?," *Computer Decisions,* Vol. 6 No II, November 1974, pp 20-23.

PART III

INTERACTION WITH SOCIETY



Area Director:
Gopal K. Kapur
Kapur and Associates
Pleasanton, California

Education—Curricula—Training

During the past year interest and controversy in computer science and information science education and training has increased notably. The keynote addresses given by George Glaser at the 1974 National Computer Conference and the 1974 EDUCOM in Canada have served to heighten this interest and controversy. Mr. Glaser's views stimulated an immediate controversy that has not yet subsided, as was demonstrated during the education seminars at the 1974 ACM conference in San Diego. The courses presently being offered in colleges and universities have been both attacked and defended vigorously.

We take the position that some good work is being done in the colleges and universities, but there is a need that can only be provided for by taking an in-depth critical look at the present curricula coupled with cooperation by the industry in defining its needs. The four seminars offered reflect this concern and address to the problems faced by the educators and the users.

The seminar, Undergraduate and Graduate Education in Computer Science, summarizes the good work being done in the colleges and universities. To balance the viewpoints presented in this seminar, another seminar, Data Processing Education—A View from Education—An Appraisal from Industry, presents a critical analysis of the existing data processing education while balancing this with the viewpoints of the educators. A third seminar, Computer Science Education for Majors of other Disciplines, has been designed for noncomputer professionals whose activities require them to interface with computers. Areas covered include biologists, chemists, physicists, sociologists, ecologists, business administrators, linguists, architects, and teachers. The fourth seminar, Use of Computers in Education, under the sponsorship of AEDS, examines the use of computers in the educational process in primary and secondary schools, junior colleges and universities.

Realignment of objectives in information system degree programs

by J. DANIEL COUGER
University of Colorado
Colorado Springs, Colorado

INTRODUCTION

Development of degree programs in information system analysis and designed lagged demand by nearly five years. Now the growth of these programs appears to be outdistancing demand.

Today's conditions dictate emphasis on quality of output rather than quantity. Considerable attention has been paid to content of curriculum and should continue, in such a dynamic field.

Pedagogical techniques deserve equal attention. With the decrease in pressure to produce large numbers, academicians can reevaluate and strengthen teaching methods. This paper will review growth projections, existing degree programs and teaching methodology.

DECLINE IN DEMAND

In their recent study, Gilchrist and Weber[1] showed that previous estimates of demand for programmers has been overstated. Three national studies[2] had forecast 1970 programmer population at levels ranging from 200,000 to 650,000. Bureau of Census studies in 1970[3] (a 20 percent sample of the U.S. population) extrapolated a programmer population that year of 161,377. Data from Labor Department Area Wage Surveys of 1970 were extrapolated by Gilchrist and Weber, resulting in an estimated population of 158,000. They surmised, "this is in surprisingly close agreement with the Census and suggests that the 'true' figure is probably close to 160,000."

The researchers then extrapolated programmer population as shown below:

Estimated Employment of Computer Programmers by Year for 1969 to 1973 Based on Area Wage Survey Data

Year	*Number of Programmers*
1969	150,000
1970	160,000
1971	170,000
1972	180,000
1973	180,000

Gilchrist and Weber concluded: "A consequence of our conclusion should be careful appraisals by educational administrators in both public and private sectors as to whether the large number of courses aimed at training programmers is justifiable on a vocational basis.

"Fewer employment opportunities than had been forecast caused a very significant drop in the number of so-called 'Private EDP Schools,' and we wonder whether similar overproduction problems may not face the Junior and Community Colleges next as there has been a considerable increase in the number of data processing courses in those institutions."

INCREASE IN SYSTEM ANALYST DEGREE PROGRAMS

Although my literature search did not reveal recent projections on demand levels for system analysts and computer scientists, the slowdown in the market for these positions over the past two years may be indicative of an environment similar to that of programming.

Data exist on the increase in degree programs to produce system analysts. A 1974 survey by the *Computing Newsletter for Instructors of Data Processing** revealed that 38 schools** have implemented degree programs in business data processing. Another nine schools plan to implement the program before 1976. The characteristics of implemented programs are revealed in the following tables.

Thirty-eight schools have implemented degree programs to prepare persons to design and implement computer-based information systems. As Table I shows, the titles of these programs vary considerably. However, the title "Information Systems" predominates.

The size of the Information Systems program ranges from five to 210 students at the bachelors level, one to 105 at the masters level and one to 20 at the doctorate level. Table II shows ranges of FTE students by program. It also depicts the number of programs by level: 28 bachelors level, 21 masters level and 11 doctorate level.

The average program size is shown in Table III: BS (76.4 FTE), MS (24.2 FTE) and PhD (8.6 FTE). Faculty size varies from one to 15; median size is 3.5 FTE faculty. Table III shows the faculty/student ratio.

* October, 1974, p. 1-8 (Box 7345, Colorado Springs, CO 80933).
** List of schools provided in Appendix I.

TABLE I—Titles of Degree Programs
(38 Schools Reporting)

Title	Frequency of Use
Information Systems	8
Business Data Processing	4
Computer Science	3
Management Information Systems	2
Information Sciences	2
Business Information Systems	2
Other (None Identical)	17

TABLE III—Information Systems Faculty-Student Ratios
(34 Schools Responding)

Degree	Mean FTE Students	Students per Faculty FTE
BS/BA	76.4	16.9
MS/MBA	24.2	4.5
PhD/DBA	8.6	2.9

As might be expected of new programs, curricula vary in proportion to level of program. The 28 bachelors-level programs are quite similar in content, fairly closely relating to the model below, recommended last year by the A.C.M. (Association for Computing Machinery).[4] The masters level programs more closely resemble the MBA option than the general program of the graduate level recommendations of the ACM.[5] PhD programs vary so much that a model cannot be approximated.

RESPONSE TO DECLINING DEMAND/INCREASING SUPPLY

The "golden era" of bountiful demand for data processing professionals appears to have suffered a premature demise. It appears that the profession will exhibit the stable, yet undramatic growth of its sister professions in the technical field.

However, there is little need to view this trend pessimistically. Rather than concentrate on quantity, academia at all levels can now emphasize quality.

Dr. Hans Hansen, in his introduction to a paper I presented in October at Wildbad, Germany, commented, "The Americans have been as interested in the approach to teaching as in the content of their teaching." I was the only American in attendance, speaking on the approach to teaching Production Information Systems in university-level curricula. Dr. Hansen reiterated comments I'd heard many times in overseas travel.

Are we overly concerned with pedagogy? A colleague at UCCS says that our job is "solely to impart the truth—whether the students partake or not is their concern." A previous member of our faculty stated it a little less eloquently, "The hogs don't care how they're slopped—just throw it to them and let them fight over it."

To the contrary, most faculty I know place importance on effective teaching methods. They are interested in both teaching effectiveness and teaching efficiency.

In a recent conference in Mexico a fellow speaker introduced his presentation by criticizing some colleagues in his profession who lectured from notes "yellowed with age." He then proceeded to use transparencies which had been used for so long that the contents were difficult to read—they were "browned with age."

Nor are those of us who use computer technology immune to pedagogical obsolescence. For one of my courses, I'm using forecasting programs which should have been improved before assignment to the students again this year—if I continue to procrastinate, the FORTRAN coding sheets in the program documentation may begin to yellow.

TABLE II—Number of Students in I.S. Program
(38 Schools Responding)

Range of FTE Students	Number of Schools BS/BA	MS/MBA	PhD/DBA
1 - 10	1	6	8
11 - 30	6	12	3
31 - 50	7	1	
51 - 75	3		
76 - 100	3		
101 - 150	5	1	
151 - 200	1	1	
Over 200	2		

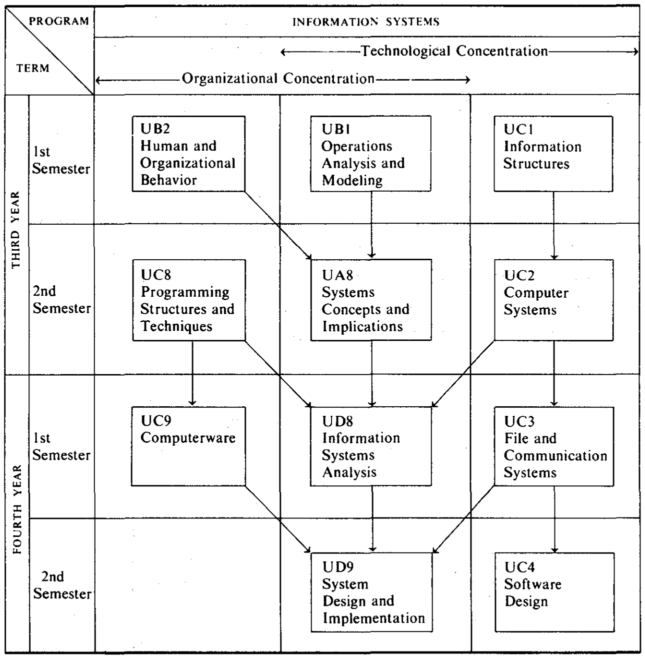

PROGRAM
TERM
INFORMATION SYSTEMS
Technological Concentration
Organizational Concentration
THIRD YEAR
FOURTH YEAR
1st Semester
2nd Semester
1st Semester
2nd Semester
UB2 Human and Organizational Behavior
UB1 Operations Analysis and Modeling
UC1 Information Structures
UC8 Programming Structures and Techniques
UA8 Systems Concepts and Implications
UC2 Computer Systems
UC9 Computerware
UD8 Information Systems Analysis
UC3 File and Communication Systems
UD9 System Design and Implementation
UC4 Software Design

Figure 1—ACM recommended undergraduate option in information systems

Unilateral emphasis on content

On the other hand, we may frequently update content of our courses and neglect the means by which that information is transmitted to students. Pedagogical approach should be reevaluated as frequently as content. Some instructors might appropriately be labeled by their students as demagogues rather than pedagogues. Demagogical pedagogues structure the learning process around themselves rather than design a multiprong instructional approach that utilizes the best teaching medium for each topical area.

For example, my experience is that students grow to understand forecasting techniques best by utilizing computer-based forecasting models. Our students learn intragroup dynamics best through application in teams playing our management game. However, portions of the statistics subject area require a combination of lecture and exercises to effectively instill both concepts and techniques.

Improving effectiveness and efficiency

Some academicians say that increased class sizes, imposed by the administration, exclude any teaching approach except lectures. I disagree. Examples to the contrary in the computer-assisted teaching programs have been developed at Florida State by Professor William Shrode (data programming instruction) and at the University of Illinois/Chicago City College by a team headed by Professor James McKeown (accounting instruction). These projects proved that both learning effectiveness and instructor utilization were enhanced despite increased class size.

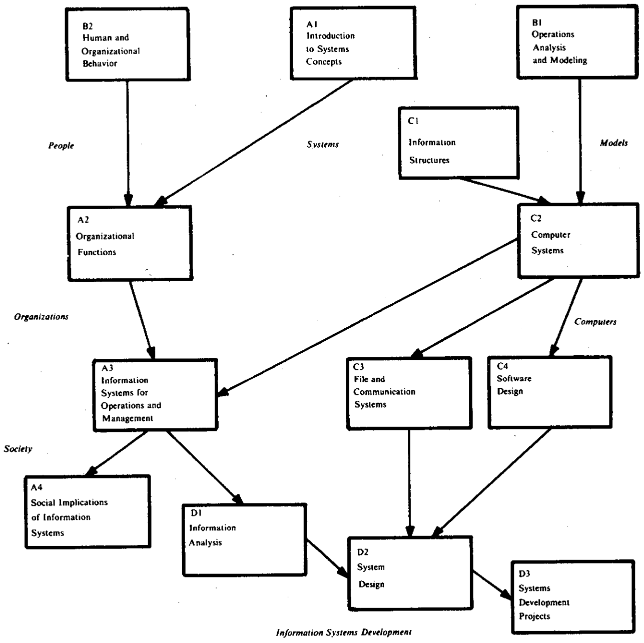


Figure 2—ACM recommended graduate program in information systems

Other instructional approaches

However, C.A.I. is not the only alternative. In the UCCS information systems course we make extensive use of video instruction. For example, students in my systems analysis and design course obtain 55 percent of their course credit for study of Deltak's multi-media (video, audio, printed materials) course outside class. They get their state-of-the-art knowledge through this means. In class discussions I can concentrate on concepts/techniques and future state-of-the-art subjects. We cover approximately 25 percent more material than was covered prior to availability of the multi-media course.

Another example of teaching innovation is illustrated by VIEWIT. Many compiler construction courses do an excellent job of analyzing each phase but fail to "put it all together" in the student's mind—the whole as the sum of individual compiler phases is not easily understood. Texts suggest that the student should write his own compiler to best understand the process, but limitations of time and resources frequently make this impossible. To use an existing compiler as a model is not generally satisfactory, since understanding and compilation efficiency are at cross purposes with one another. The VIEWIT compiler was designed by Professors E. M. Henshon and N. E. Sondak (Worcester Polytechnic Institute) to serve as a model for the compiling process. It supports a modified version of the BASIC source language which offers reasonable processing power while being similar enough to keep the compiler design straightforward. A command language is supplied to allow the user to display the process which VIEWIT performs on the source program input. The VIEWIT compiler is implemented in FORTRAN. In addition, the various functional routines of VIEWIT are highly modular in construction so that they can be easily replaced to allow some sophisticated algorithms to be used in the compilation or to allow extensions to the original source language.

CONCLUSION

With the relief in pressure to produce large numbers of graduates, energy can be directed toward improving the quality of our programs.

The dynamic characteristics of our field force a continuous reevaluation of curriculum content. We need to be equally concerned with our teaching methodology. Just as the cobbler's child is the last to be shod, we are often the last to use the computer to enhance our teaching.

REFERENCES

1. Gilchrist, Bruce and R. E. Weber, "Enumerating Full-Time Programmers," *Communications of the ACM*, October 1974, pp. 592-593.

2. *The State of the Information Processing Industry*, AFIPS Press, Montvale, N.J., 1966.
Gilchrist, Bruce and Richard E. Weber, "Employment of Trained Computer Personnel—A Quantitative Survey," *Proc. AFIPS, 1972 SJCC*, Vol. 40, AFIPS.
Occupational Outlook Handbook, 1970-71, Ed. U.S. Dept. of Labor, Washington, D.C.
3. *Detailed Occupation of Employed Persons by Race and Sex for the United States*, 1970, U.S. Dept. of Commerce, Bureau of the Census, Washington, D.C., 1973.
4. Couger, J. D., ed., "Curriculum Recommendations for Undergraduate Programs in Information Systems," *Communications of the ACM*, December 1973, pp. 727-749.
5. Ashenhurst, R. L., ed., "Curriculum Recommendations for Graduate Professional Programs on Information Systems," *Communications of the ACM*, May 1972, pp. 364-398.

APPENDIX—LIST OF AACSB SURVEY PARTICIPANTS WITH INFORMATION SYSTEMS PROGRAM

School	*Address*
University of Alabama	University, Alabama 35486
Arizona State University	Tucson, Arizona 85721
University of Arkansas	Fayetteville, Arkansas 72701
The Bernard M. Baruch College	17 Lexington Ave., New York, New York 10010
Boston College	Chestnut Hill, Massachusetts 02167
Bowling Green State University	Bowling Green, Ohio 43403
University of California	Berkeley, California 94720
University of California, Los Angeles	Los Angeles, California 90024
University of Colorado	Colorado Springs, Colorado 80907
California State University, San Francisco	1600 Holloway Avenue San Francisco, California 94132
California State University, Sacramento	6000 Jay Street Sacramento, California 95819
Case Western Reserve University	Cleveland, Ohio 41106
Dartmouth College	Hanover, New Hampshire 03755
University of Delaware	Newark, Delaware 19711
Duquesne University	Pittsburgh, Pennsylvania 15219
Eastern Michigan University	Ypsilanti, Michigan 48197
University of Florida	Gainesville, Florida 32307
Fordham University	Bronx, New York 10458
University of Georgia	Athens, Georgia 30602
Harvard University	Soldiers Field, Boston, Massachusetts 02163
University of Iowa	Iowa City, Iowa 52242
Kent State University	Kent, Ohio 44242
Louisiana Tech University	Box 5796, Tech Station, Ruston, Louisiana 71270
University of Minnesota	Minneapolis, Minnesota 55455
Mississippi State University	Mississippi State, Mississippi 39762
University of Nevada	Reno, Nevada 89507
New Mexico State University	Las Cruces, New Mexico 88003
Northern Arizona University	Flagstaff, Arizona 86001
Northern Illinois University	DeKalb, Illinois 60115
Northeast Louisiana University	Monroe, Louisiana 71201
Ohio State University	Columbus, Ohio 43210
Purdue University	West Lafayette, Indiana 47907
The University of Rochester	Rochester, New York 14627
Texas A&M University	College Station, Texas 77843
Texas Tech University	Lubbock, Texas 79409
Temple University	Philadelphia, Pennsylvania 19122
Tulane University	New Orleans, Louisiana 70118
University of Southern California	University Park, Los Angeles, California 90007
Washington University	St. Louis, Missouri 63130
Washington State University	Pullman, Washington 99163
Western Michigan University	Kalamazoo, Michigan 49001

Undergraduate programs in computer science

by NORMAN E. SONDAK
Worcester Polytechnic Institute
Worcester, Massachusetts

Since the publication of "Curriculum '68", that landmark report has been subject to numerous discussions, enlargements, and criticisms. An updated version of this work is currently under preparation. Some of the more significant criticisms which were directed at the original report concern the lack of attention to the physical aspects of computer science, the limited consideration of data processing as a profession, and the absence of courses in the area of computers and their impact on society. The computer science graduate and undergraduate programs at Worcester Polytechnic Institute were developed and implemented over the same time frame as "Curriculum '68". While the departmental computer science academic program was developing, Worcester Polytechnic Institute itself was simultaneously engaged in an exciting and unique experiment in undergraduate science and engineering education entitled the WPI Plan.

The experience in this unusual milieu has served as a basis for the computer science undergraduate program now in effect at WPI. It is believed that the ideas generated by this symbiosis of a new academic discipline along with an unusual approach toward science education can be successfully incorporated in other computer science undergraduate programs to broaden and deepen the undergraduate exposure to computer science as a profession.

The WPI Plan stresses the attainment of the Bachelor's degree through the demonstration of competence while allowing a broad freedom in course selection supported with academic advisory assistance. The basic B.S. degree requirements at the Institute are, along with course work, completion of a competency examination in the major field of study, qualification in a minor field of study (normally in the humanities), and two projects, one of which is in the major field, and the second, preferably one relating technology to society. As can be seen, the framework of the WPI Plan has led the Computer Science Department to quite naturally supplement some of the major shortcomings voiced about "Curriculum '68".

The project activity has been particularly beneficial. It gives the undergraduate computer science major a perspective on the real world of computing. The department has a number of projects developed in conjunction with local industry. In this respect, WPI is quite fortunate to be in close geographic proximity to a number of major computing equipment manufacturers and large computer users, such as, Digital Equipment Corporation, Data General Corporation, Sanders Associates, and Honeywell Information Systems Division, as well as Norton Company and State Mutual Insurance Company.

Actual projects have ranged from software verification to contributions to the design of a computer based information system for the juvenile court of Worcester. Student reaction to the projects both of the on and off campus varieties has been uniformly excellent. The experience of organizing and executing the computer science activities required is now of a much larger scope than normally encountered in regular course work and reaction by students is extremely positive. In addition, the exposure to various employers of computer science students is a fortunate side-effect for students and often leads to offers to join that firm upon graduation. Several of our outstanding students have been hired by local computing equipment manufacturers or computer application firms as a direct result of their project activity.

The requirement to produce and administer a competency examination in computer science has forced the department faculty to come to grips with a very difficult problem: defining the essence of computer science in such a manner that an undergraduate student's competency in this field can be quantitatively measured. This is difficult in any academic field of endeavor, but even more so in an area such as computer science with its dynamic growth and broad areas of applications.

To accomplish this goal students are usually allowed to select one of three problems. These problems are designed to permit a broad latitude in the actual solution. The student can bring to bear wide ranges of knowledge and experience in developing the written answer to this portion of the competency exam. The second phase of the examination is an oral review. Here the student explains his written solutions or enlarges it after he has time to think about it. In addition, other concepts of computer science may be examined at this time. Our experience with the competency examination has been gratifying. The students usually think of it in advance as a traumatic experience, an obstacle to be overcome. But after taking the examination, the general comment has been that the examination was fair and helped crystalize many aspects of the profession of computer science that they had not thought about previously.

The need for one to one advising as required by the WPI

Plan during the development of the undergraduate program for each student has also caused each member of the faculty to be intensely interested in the general aspects of an academic program in computer science, as well as employment opportunities or graduate school availability.

In total then, the unusual and innovative aspects of the WPI Plan along with the basic core curriculum have caused our students to receive a much more individualized and broader education in computer science than the traditional lock-step program might have allowed.

From the employment point of view, the results have been also positive. We have a limited base as far as numbers of graduates from the Plan are concerned, however, those students graduating to date have had little difficulty obtaining the types of jobs they have wanted. In general, they have had several offers to select from. Thus far, most of the students have gone to work with equipment vendors in the local area such as Digital Equipment, Data General, and Honeywell. But a significant number are beginning to join major insurance, banking, and other financial institutions as well as manufacturing plants in the region.

The practical project activity, a careful and individualized advising system and a continuous examination and concentration by the department on what they consider to be the essence of computer science is certainly transportable to any computer science undergraduate program. This concern and commitment by computer science departments should improve the probability of employment for their graduates and help to effectively answer a number of criticisms leveled at current computer science programs by employers.

Graduate education in computer science and its relationship to industry

by M. C. YOVITS
The Ohio State University
Columbus, Ohio

There are a number of different focal points which characterize various graduate programs in computer science. Some of these programs are theoretical and mathematically oriented. Some programs are organized around management or business applications, some around computer systems, some around information storage and retrieval, and so forth. Many examples of these and other orientations for graduate computer science programs can easily be provided.

At Ohio State we have decided to build a broad base for our various programs in computer and information science, the name we have chosen as most descriptive of our program at Ohio State. This broad base encompasses most of the various possible focal points for computer science. It is only with such a broad interrelated set of programs and objectives that graduate students can be adequately educated to solve general industrial types of problems. Furthermore, the breadth of the programs and faculty makes for an exciting and dynamic environment not otherwise found.

We have developed fourteen fields of specialization which our Ph.D. students can use either as major or minor areas of specialization. These are:

1. General theory of information.
2. Information storage and retrieval.
3. Theory of automata and theory of computation.
4. Artificial intelligence.
5. Pattern recognition.
6. Computer programming, including system programming.
7. Theory and processing of programming languages.
8. Digital computer architecture and organization.
9. Numerical analysis.
10. Man-machine interaction and systems.
11. Formal and computational linguistics.
12. Management information and systems.
13. Biological information processing.
14. Social, economic, and psychological aspects of information production and processing.

Students can also choose minor areas of specialization from other appropriate departments having common interests with the Department of Computer and Information Science.

Ohio State University is a large, diversified university, and furthermore it is the only State University in Ohio which gives a wide range of Ph.D.'s. Accordingly, the Department of Computer and Information Science has a sizable graduate program of slightly less than 200 students. Roughly 75 percent of these graduate students enter the program with the objective of earning a Master's degree and then leaving to take employment. We consider these students to be professionally oriented and thus consider our degree program at the M.S. level to be primarily a professional degree. The total number of students who have been graduated with Master's degrees now number well in excess of 250. These graduates have taken employment, mostly in systems type of work in one form or another, with many different industrial, governmental, and non-profit organizations around the country and even abroad.

The Master's students currently have seven different options which they can pursue and which define their objectives. These are:

1. Theoretical Foundations
2. Information Systems
3. Computer Systems
4. Numerical Analysis
5. Operations Research
6. Biomedical Computing and Information Processing
7. Administrative Science

All students, regardless of the option selected, are required to take a set of courses chosen from a prescribed core. The student then completes his course of study with about 15 quarter hours of electives from a group of courses specific to each particular option. Courses in the core include:

1. Mathematical Foundations of Computer and Information Science
2. Advanced Seminar in Computer and Information Science
3. Principles of Man-Machine Interaction

4. Numerical Analysis
5. Advanced Computer Programming
6. Digital Computer Organization
7. Introduction to Linguistic Analysis
8. Modern Methods of Information Storage and Retrieval

From these, with certain restrictions, the student must choose 29 quarter hours. In addition, the student is required to take 10-15 hours of mathematics and statistics.

The Master's degree program is of considerable interest to industrial organizations. From conversations which I have had with recruiters and technical managers, I am told that our students perform well. What is more important, the industrial organizations continue to come back year after year asking for more of our Master's students. In general, our Master's students have had little difficulty finding the job of their choice. Graduates who visit the campus after working for a period indicate that they are well pleased with their education and feel that generally it has been good preparation for their jobs.

In a recent poll of our Master's graduates, we found that 59 percent of our graduates felt that their backgrounds were superior to those of their colleagues and 36 percent felt that they were comparable. Further details on this poll may be found in the reference by Kerr and Kalmey.[1]

At the same time, we also have a large Ph.D. program with a steady state output of about 15 per year. Although as we know academic positions are now plentiful in computer science, I feel that this situation will change in the near future as universities undertaking new and expanded computer science programs complete their staffing. Accordingly, we recognize that most of our Ph.D. graduates must expect to find employment in industry or government. Thus far, in the three years that we have been awarding Ph.D.'s approximately 50 percent have taken non-university positions.

We believe that the type of broad program which we have developed in computer and information science with a good fundamental component, yet with considerable involvement with applications, is of particular interest to industry. Our students know the basic fundamentals and are, as well, involved with and understand many applied problems and general applications. These students generally have the breadth and the understanding to solve problems. This is the type of graduate that industry is looking for. What is more, they should be prepared to solve many different types of problems, not only narrowly defined problems.

One of the main ways that a program such as ours can interface with industry is through the recruiting process—our graduates will be hired by industry. However, in a relatively large urban setting there are many other ways to establish relationships. For example, a number of our students are able to find part-time employment in local industrial and non-profit organizations while they are studying. In many cases we are able to fill these jobs through the Department as a type of graduate assistantship. In some cases, these part-time positions lead to full-time jobs after graduation. Furthermore, additional interactions take place through employees of many industrial organizations in the area who enroll in our graduate program.

Other interactions which take place with industry include consulting on the part of our faculty and the use of adjunct faculty from industry to help us teach some of our courses. The adjunct faculty are important to the educational program in a number of different ways, but there are two in particular that we feel are most important. Foremost is the fact that adjunct faculty can bring to our students the real industrial problems and we try, where it is possible and mutually agreeable, to develop courses for the adjunct faculty that emphasize these problems. It is primarily through courses of this type that our students can learn about the real world. Unfortunately, because of their responsbilities in their own organizations, it is not always possible to find available faculty who are able and willing to teach on a regular basis.

A second way in which adjunct faculty from local industry can be very helpful is in helping to staff our evening program. Many of our students who are employed in local industry find that it is most convenient to take courses in the evening. Accordingly, we have developed a growing evening program and are able to use adjunct faculty from local industry to help staff this program. This arrangement turns out to be mutually satisfactory to all parties concerned. The adjunct faculty are generally more available in the evening.

Finally, other important relationships which are most useful in fostering an interaction are: inviting industrial people to present colloquium talks on current areas of interest and developing short courses for industry as the need occurs either on the university campus or at the industrial site.

In summary, we believe that the interaction with industry is essential for both our Master's and our Ph.D. students. We believe that preparation for industrial positions is essential for most of our graduate students, and a part of our curriculum is geared in that direction. A broad and dynamic graduate program enhances this type of an education. Further interactions of importance both to the university and to industry take place through the use of adjunct faculty to teach courses and the use of our faculty for consulting. Enrollment in our courses of students from local industry is another important catalyst.

REFERENCE

1. Kerr, D. S. and D. L. Kalmey, "The Bachelors and Masters Computer Science Graduate," Submitted to the *2nd World Conference on Computers in Education*, 1-5 Sept., 1975.

The role of continuing education in computer science

by WALTER J. KARPLUS
University of California
Los Angeles, California

In the present context, "Continuing Education" refers to formal courses offered to active professionals for purposes of career development rather than with an academic degree objective. The general need for continuing education in a rapidly evolving field such as Computer Science is so well accepted as to require no further justification at this time. Rather, it is the purpose of this paper to consider the diversity of approaches to advanced education outside the conventional academic framework and to comment on current attitudes and trends.

FORMAT

The overwhelming majority of participants in Continuing Education programs are employed on a full-time basis either in industry or by government agencies. Usually the employer contributes most if not all of the tuition and expenses incurred by the student. It is necessary therefore that the format of the Continuing Education program take into account the needs and preferences of the employer as well as of the student, particularly as they relate to time lost from work. There appear to be three basic approaches to the scheduling of Continuing Education courses.

Periodic lectures

The student attends formal presentations either weekly or bi-weekly over a period corresponding to a college semester or quarter. Frequently, the material offered in such courses corresponds closely to the material offered during the daytime to regular college or university classes, and occasionally credit earned in such courses can be used toward academic degrees.

Short courses

The student attends an intensive series of lectures approximately seven hours per day for a consecutive period of days. Some short courses last only two days while others span a two-week period. Such courses usually bear no direct resemblance to academic offerings and are frequently taught by "stars" in their technical specialty.

Retreats

A group of students are housed at a relatively remote location for a period of one to four weeks and receive intensive instruction, approximately five or six hours per day, from a battery of lecturers. The formal lectures are generally supplemented by evening programs and workshop sessions.

LOCATION

Depending upon the subject matter and the demand, Continuing Education programs are presented at a variety of sites.

College campus

Most Continuing Education programs are offered on college or university campuses, taking advantage of available classroom, library, and computer facilities. For many colleges this constitutes a convenient way of utilizing their buildings during the evening hours and during vacations; some universities have allocated special classrooms and buildings exclusively for Continuing Education programs.

In-plant

Many large companies have found it expedient to offer Continuing Education programs for their own employees either in their own facilities or nearby. These courses are usually taught in part by specialists on the staff of these companies and in part by imported lecturers.

Road-show

One or a group of lecturers presents essentially the same short course at a number of locations throughout the country and occasionally overseas. Usually, these programs last two or three days and are offered in large urban centers such as New York, Washington, D.C., Los Angeles, etc.

MOTIVATION

Most students enroll in Continuing Education programs in order to advance their professional career. Depending upon the student's specific job responsibilities and upon his maturity, different approaches to the subject matter may be appropriate. Some Continuing Education programs try to strike some sort of balance between approaches so as to accommodate a broader range of participants. Usually, however, it is possible to identify one of the following as being the major objective of the presentation and the major motivation for the participant.

Depth

The purpose of such a program is to present the latest advances in a highly specialized area. The student should have considerable familiarity with the general subject and desire to learn of the latest advances in the field. The instructor, usually a well-known authority in a specialty, supplies copious notes including information not readily available in the literature. The short course format is the most logical vehicle for such a course, and numerous universities and many private organizations offer such programs either on campuses or on a "road-show" basis. These courses are pitched to various levels of professionals, but most emphasize depth in exploring highly specific issues. During the academic year 1974–75 UCLA, for example, is offering over thirty short courses in the Computer Science field alone. As described in more detail elsewhere,[1] approximately one half of these courses are taught by university professors while the other half is offered by recognized authorities in industry. Most of the courses run for five consecutive days, although there are a few three-day and a few two-week programs.

Breadth

Here the student generally desires to round out and at the same time update his technical education by taking college type courses. Most frequently, the material offered in the course is only indirectly related to his present work assignment, and the student looks to the course as an aid to changing his technical specialty or to advance himself within his organization. Although some of these courses may manifest a mathematical depth considerably greater than that found in most short courses, the general approach taken by the lecturer is to provide complete upon the latest advances. UCLA offers approximately twelve such courses in the Computer Science area each academic quarter. Some of these are introductions to programming, but most correspond to courses offered to juniors and seniors in Computer Science during the regular school session. All of these courses are presented using the "periodic lecture" format, with class meetings of two to three hours every week for a twelve-week term. In general, the teachers of these courses are less distinguished than those offering short courses, and the tuition on a per hour basis is less than one-third of that for most short courses.

Overview

Here the audience is comprised primarily of middle and upper level managers and executives. Usually the student will have earned a Bachelor's or Master's degree in a technical area fifteen to twenty years earlier; but because of job pressures he has been unable to remain up-to-date and to explore newly-emerging technical disciplines. Such a student will expect to be briefed on the significance of new developments and to obtain a perspective over the interrelation of currently fashionable devices and techniques. He is not interested in absorbing a large number of facts or in learning new skills. For this purpose the "retreat" format appears to be the most suitable. The manager physically absents himself from his job for a number of weeks and essentially emerses himself in a carefully planned and coordinated program. For many years, UCLA offered a program of this type entitled "Modern Engineering for Engineering Executives". Lasting from four to six weeks the program was offered in a resort-like location to a class of approximately thirty executives. More recently, a similar program has been offered on a private basis to executives and managers of companies such as IBM, Boeing, Texas Instruments, and a number of others.

CURRENT TRENDS

Fifteen years ago most Continuing Education programs were of the "periodic lecture" variety. Today the short course format appears to be the most widely accepted. From the student's point of view a short course constitutes a welcome change of pace from his regular work routine—an opportunity to mingle briefly with his peers from other companies, and a stimulus to engage in self-study. From the point of view of organizers of Continuing Education programs, a short course usually attracts a higher level and therefore a more affluent audience, and the compact scheduling of the lectures facilitates the obtaining of top-notch lecturers. The latter usually find short course programs far more lucrative than any other teaching opportunity. By contrast, the students usually find that a course offered along the "periodic lecture" format comes as an addition to his day-to-day responsibilities, necessitating one evening of attendance and at least one additional evening of self-study per week. Such a course therefore requires a much longer-lasting and serious motivation than the short course. As a result, attendance at "periodic lecture" programs appears to be diminishing, while short course programs are on the ascendance.

Another current trend is toward formal recognition of Continuing Education programs. A number of universities, including UCLA, offer special certificates for the completion of a specified number of Continuing Education courses. Throughout the nation the concept of the Continuing Education Unit, (CEU) is gaining increasing acceptance. One CEU is earned for every ten hours of Continuing Education class contact and provides a formal record and recognition of the student's participation in Continuing Education programs.

REFERENCE

1. Ingersoll, Alfred C., "Continuing Education and The Computer Scientist," *The UCLA Computer Science Department Quarterly*, Volume 2, Number 4, October 1974, pp. 13-20.

The role of computer science minors in undergraduate and graduate curriculums

by GERALD N. PITTS and BARRY L. BATEMAN
Texas Tech University
Lubbock, Texas

It has been estimated that over 200,000 computer related jobs went begging in the United States in 1974 because of lack of qualified college graduates. Both Industry and Government are concerned about the lack of practical knowledge of college graduates. A national survey estimated that 95 percent of all undergraduate business majors had to be re-educated to the tune of $8,000 each on the average before they could be considered productive workers.

Many academic disciplines such as business, engineering, mathematics, etc., have recognized this problem and have begun to offer a limited number of computer-related courses in each discipline. These attempts have not been fruitful because of the lack of computer science expertise of the instructors. This is to be expected because a professional cannot be an expert in every academic field. In other words, engineers normally do not attempt to teach English, Biology, etc., in the Engineering discipline. It therefore seems reasonable to develop computer science minors for related disciplines, administered by the computer science discipline exclusively.

Many universities have developed computer science major disciplines, but few have concentrated on the need for computer science minor programs since the primary effort for most academic discipline developments is in the major course areas. Much effort goes into cultivating students with majors in computer science (in some cases "luring" students from related disciplines). Some generalized courses (non-major) are sometimes created to boost student credit-hours for the department such as introduction and survey courses. This type of protective development further alienates related disciplines from computer science.

Courses should be developed with specific related disciplines in mind for providing the greatest benefit of computer knowledge to that discipline. This type of course development (minor in computer science) such as developed at Texas Tech University, Lubbock, Texas, is inherently interdisciplinary. A program of approximately 18 semester hours of computer science has proven to provide a sufficient base of computer science knowledge. A core of courses that include the most commonly used computer languages (FORTRAN, COBOL, BASIC, ASSEMBLY) should be taken before more specific discipline-related courses are taken. The languages provide a basic understanding and appreciation for computer processing, logic, and applicability. Specific applicability can then be provided for each separate discipline.

Not all computer-related jobs require the broad range of computer science knowledge which is normally provided to computer science majors, but basic computer knowledge applicable to the application area is of use. Students minoring in computer science at Texas Tech are encouraged to engage in actual practical application of computer knowledge of their particular discipline. Local industry has an indepth appreciation of the value of the computer science minor program at Texas Tech and are involved in hiring these students as well as providing feedback to the department about lacking areas. The Texas Tech Medical School utilizes a good many mathematics and engineering majors for computer operations, maintenance, programming and systems analysis. Several manufacturing firms utilize a large number of business majors for computerized accounting, inventory control, and production control program development and maintenance. Computer science minors in an interdisciplinary environment should be considered as a viable alternative to computer science training.

Computer science education for majors of other disciplines

by J. A. ARCHIBALD, JR.
State University of New York
Plattsburgh, New York

INTRODUCTION

There is an old saying that a specialist is a person who "knows more and more about less and less." It would be nice to believe that that saying, if ever true, is no longer true. Our world does not consist of little, isolated problems and situations which exist totally independent of their surroundings; it consists, rather, of a massive set of interconnected and interrelated objects and events. While, on the one hand, practical considerations make it necessary to isolate subsystems for the purposes of conducting detailed study, on the other hand a subsystem cannot be fully understood without understanding its interfaces with and relationships to its environment and the other systems with which it is associated. Indeed, we are witnessing the beginnings of this type of interdisciplinary activity with the present levels of interest and trends in biochemistry, biophysics, and biomathematics,[1] as well as in some of the modern approaches to the study of the environment. Thus, we recognize that educational programs can no longer address themselves to single, isolated disciplines. Indeed, the approach to higher education needs to be interdisciplinary to a degree never before appreciated. The present need is not for a scientist with a monolithic perspective who has majored in a traditional discipline in a traditional manner, but rather a scientist with an interdisciplinary perspective who has studied in a broad, interdisciplinary program with an area of specialization. Computer Science must contribute, effectively, to broad, interdisciplinary programs for individuals specializing in a wide variety of fields. We must address ourselves to two primary considerations with equal vigor: the development of specialized understandings for the practice of Computer Science, and the development of generalized understandings to aid in the practice of other disciplines. We must present a degree program that includes sufficient study in the other disciplines to enable the new practitioner to participate effectively and innovatively with professionals from other disciplines in the solution of problems, and we must provide sufficient instruction to majors in other disciplines that they will be able to understand the relationships between the respective disciplines and the role and utilization of Computer Science in the solution of problems in their disciplines. In both cases, more emphasis needs to be placed upon interfaces. The need for such action has been mentioned many times in the past.[2,3] The responsibility of providing appropriate programs for majors was commented upon in an earlier paper.[4] The second responsibility, Computer Science for majors of other disciplines, is discussed herein.

THE NATURE OF COMPUTER SCIENCE

Before we can really discuss the question of how Computer Science interfaces with other disciplines (and therefore how Computer Science Education interfaces with education in other disciplines), we must get some idea as to just where the interfaces are. In some manner we must come to grips with exactly what is Computer Science. Regretfully, there is no universally accepted definition. In what follows, we will not supply a definition either, but rather present an indication of breadth.

Professor Knuth has suggested that Computer Science is the study of algorithms.[5] This suggestion is, regretfully, too narrow. Computer Science certainly includes the study of algorithms. It also includes the study of problem solving, the study of information, the application of algorithms to the solution of problems, and the design and utilization of devices capable of using algorithms to solve problems. Within the concept of the "study" of algorithms, we must certainly include the design, representation, translation, and (design and use of languages for) communication of algorithms. Within the concept of the "study" of information, we must certainly include the nature and design of information structures, as well as the meaning of information.

There are some who will argue that Computer Science is a wholly contained subdiscipline of some other more conventional discipline, like mathematics or electrical engineering or business administration. (These trends have, in many places, led to the inclusion of Computer Science "sub-departments" within larger departments in more conventional disciplines.) These notions should also be rejected as being too narrow. Computer Science spreads out over several related disciplines, and shares with these disciplines certain sub-disciplines that traditionally have been located exclusively in the more conventional disciplines. Within this category are such things as numerical analysis and statistics—included without the implication of removing them from mathematics.

It is also possible to recognize within Computer Science certain types of conventional splits, such as abstract versus applied Computer Science, and theoretical versus experimental Computer Science.

The point is that in the training of the interdisciplinary scientist with specializations in specific areas, broad views as to the limits of each discipline must be taken. Recognizing this, the practitioners of Computer Science, one of the newer disciplines, must be careful not to unduly limit themselves.

THE DEVELOPMENT OF AN INTERDISCIPLINARY PERSPECTIVE

A new philosophy, be it an interdisciplinary perspective, or any other type of departure, is not simply assumed by a static department—it must be planned for during a period of growth. When adapted in this manner, it influences the qualifications sought in new appointees, and gains permanence with the arrival of the new faculty. In the case of Plattsburgh State, the interdisciplinary requirements were recognized before the establishment of the Department. A committee was formed from among interested members of some of the various departments with which computing is associated. Included were the Departments of Physics, Chemistry, Mathematics, Biological Sciences, and Administrative Science and Economics, as well as the Division of Education. The perspective of this committee was that of Computer Science as a supportive discipline to their own areas of interest. A Computer Science degree program was, at this time, of only secondary interest. With this perspective clearly in view, the committee set out to search for full-time (applied) Computer Scientists to be appointed to the new Department. In the search that followed, greater emphases were placed upon professional activity in industry, including experience and accomplishments, than upon the more traditional academic activities. The two individuals who were appointed as a result of this search had a total of over twenty years of professional experience in industry, most of which was research oriented. Their formal educations, including advanced degrees, were in mathematics and physics. Their experiences were in the areas of numerical applications, nuclear physics, biophysics, physiology, and computing systems. The important point was that these individuals had an applications oriented, interdisciplinary perspective. The Department thus formed from the members of the old committee (as part-time members) and the new full-time appointees was created with the desired philosophy. All subsequent additions have reflected this philosophy. Each new appointee has come not only with adequate academic credentials, but also with strong industrial experience.

DEVELOPMENT OF THE SUPPORT PROGRAM

At the outset, our perspective was Computer Science in support of the other sciences. This perspective was heavily physics oriented, and was concerned with several general types of tasks:

1. the calculation of numerical approximations to the solution of systems of (partial) differential equations which exactly describe the activity of complex physical systems,
2. the calculation and analysis of numerical approximations of systems of (partial) differential equations which form deterministic models that approximate even more complex physical systems,
3. the development, verification, and use of stochastic (Monte Carlo) simulation models that approximate physical systems,
4. the analysis of data obtained from measurements of actual physical systems, and
5. the analysis of data obtained from the use of stochastic simulation models of physical systems.

Given these types of activities, it was obvious that our support program would need to provide certain very special items:

1. a thorough indoctrination in programming for scientific applications (i.e., FORTRAN),
2. a thorough understanding of the application of a wide variety of statistical techniques, and the utilization of established statistical procedures,
3. a thorough understanding of the applications of a wide variety of numerical methods for the solution of algebraic and differential equations.

With these requirements in mind, our first courses were established. At this point, they were intended primarily as support for students in science and mathematics—we had no degree program. Our initial courses were:

1. Introduction to Computer Science I. This is a thorough course in elementary and intermediate FORTRAN programming—intended to prepare the student to make effective use of FORTRAN in support of other disciplines, with a side objective of being able to read the FORTRAN programs of others—with no intent to include more than the bare essentials of computing fundamentals.
2. Introduction to Computer Science II. This is a course in advanced FORTRAN techniques plus a thorough indoctrination into the fundamentals of computing, and an overview of various aspects of applied Computer Science.
3. Computer Analysis of Statistical Data. This course was originally designed as a second course in statistics, concentrating on the characteristics, significance, and utilization of common statistical tests and

distributions studied through the use of experimental data.

4. Simulation and Modeling. This course concerns itself with the formulation and utilization of models which represent various kinds of systems, and includes techniques such as Monte Carlo, as well as an overview of simulation languages.
5. Introduction to Numerical Methods. This course provides an indepth study of certain numerical processes, with emphasis upon error analysis.

With these courses, we were able to contribute, positively, to the major programs of students in science and mathematics. While the original thinking had been physics oriented, the various courses were widened to include applications from other of the sciences (specifically biology and chemistry). This was the result of two things: the natural exchange of ideas among the members of the Department (which included chemists and biologists) and the assignment to our students of independent projects from their major. Actually, the latter requirement resulted in the full-time Computer Scientists being involved with faculty members from outside of the Department. We were actively serving the needs of both faculty and students in other disciplines, such as biology and chemistry. In so doing, we found not so much a need for additional course content, but rather for a broadening of illustrative problems and examples. We did place more emphasis upon such things as probability and statistics, and the application of computers to genetics, community and population studies, chemical, physical, and nuclear reaction studies, determination of physical and chemical properties, ecological systems, and molecular structure studies.

The full-time Computer Scientists also had their own perspectives widened by working with members of the Division of Education and the Department of Administrative Science and Economics, holding joint appointments in Computer Science, as well as with concerned members of the Department of Psychology. The thrusts here were primarily statistical, and, to a lesser extent, simulation. The specific interests that the Department was led to as a result of these relationships were in the areas of business applications, artificial intelligence, linguistic analyses, economic systems, and environmental studies. These led directly to the establishment of undergraduate courses in Non-Numeric Methods, Artificial Intelligence, and an Introduction to Electronic Data Processing (cross-numbered under Administrative Science). In addition, the Simulation and Modeling course was cross-numbered in the Environmental Studies program, and a graduate course was established in Computer Applications in Education (cross-numbered in the Division of Education). A summer workshop in the Simulation of Environmental Systems has also been arranged.

The program for majors of other disciplines was then completed, at the graduate level, with the institution of two introductory graduate courses, one for in-service teachers who wished to upgrade their own competencies to be able to teach computing courses in high school, and one for liberal arts graduate students who wished to use computers in research. Of necessity, as graduate courses, they contained a lot more of both breadth and depth than the undergraduate courses, including subject material from several different undergraduate courses in one package.

The final event in the development of this interdisciplinary approach toward strengthening programs for majors of other disciplines came with the canceling of the statistics offering (Computer Analysis of Statistical Data) in favor of including the content in the Statistics offerings of the Department of Mathematics, and having it offered by a joint appointee.

CONTINUING DEVELOPMENT AND PROSPECTS FOR THE FUTURE

In more recent times, we have actively sought out colleagues of other disciplines to develop, jointly with them, an understanding of how computers may be effectively used in their disciplines. We have also continued to collaborate on significant student projects. These have given us the input that we need to keep our courses relevant to the practice of these other disciplines. They have also resulted in our gaining some small degree of competence in the other disciplines—a result for which we are most grateful.

In order to support this interdisciplinary thrust, we have also kept communication lines open with people in the industrial practice of the discipline concerned. This is done by building upon our personal contacts from our industrial careers, and through participation in professional meetings catering to audience groups beyond the college and university campus.

RELATIONSHIP TO THE MAJOR PROGRAM

As stated previously, the orientation of the Computer Science Department at Plattsburgh is toward applications. Accordingly, when the major was established, it was hoped that graduates would be able to make significant contributions toward the application of Computer Science to the problems of other disciplines. Thus, the support courses described above, plus an area of concentration from a quantitative discipline, were included in the degree requirements. The package was completed by the addition of certain courses intended only for Computer Science majors, e.g., Machine Language Programming, Programming Languages, Discrete Structures, and Operating Systems. This program has already been described in detail.[4]

It is noted that a new, business oriented program, is under development.

CONCLUSION

A major, and often neglected, responsibility of any academic department, is to provide a strong program of support courses for the majors of other disciplines. If this is properly done, it can also form the basis for an applied major program. In a discipline with a high potential for interdisciplinary applications, such an approach is essential.

REFERENCES

1. Proceedings, Symposium on the Mathematics of Large-Scale Simulation, Society for Computer Simulation, 1974.
2. Gaskell and Klamkin, "The Industrial Mathematician Views his Profession: A Report of the Committee on Corporate Members," *The American Mathematical Monthly,* Vol. 81, 1974.
3. "Industry Reaction to Computer Science Education," *SIGCSE Bulletin,* Vol. 6, No. 1, February, 1974.
4. Archibald, J. A., Jr. and M. Katzper, "On the Preparation of Computer Science Professionals in Academic Institutions," *AFIPS Conference Proceedings,* Vol. 42, 1974.
5. Knuth, D. E., "Computer Science and its Relation to Mathematics," *The American Mathematical Monthly,* Vol. 81, 1974.

Data base education for students of management

by R. CLAY SPROWLS
University of California
Los Angeles, California

INTRODUCTION

One major field of study in the Graduate School of Management at UCLA is in Computers and Information Systems (CIS). This major has evolved from a single course on Electronic Computers in Business introduced in 1957 to one with more than a dozen courses devoted to some aspect of education about computers and information systems. The Announcement of the School describes the program as "designed to provide students with the basic conceptual framework and tools of analysis necessary for the design, implementation and control of information systems. These studies have the goals of training students to develop and implement management information systems—especially those using computers—for a variety of organizations both public and private. . . . In particular, the computer and the many technical advancements that have accompanied its development have had a major impact upon information systems design. Therefore, students are expected to acquire a basic understanding of computer technology and terminology and a competence in computer programming. Beyond this, students may pursue studies in computer systems analysis, management of EDP activities, design of computer-based management information systems, modeling and computer simulation, and generalized data base management systems."

The Professional Master's Program (PMP) for students majoring in Computers and Information Systems in typically a two year program. The program consists of Nucleus studies, Common Knowledge Requirements, a Concentration in CIS and Electives. The depth and the breadth of the total program are indicated by the listing of the basic studies in each part of the program.

Nucleus courses that are required of all students in the first year of the program are:

Individual Decision Making
Managerial Decision Making
Complex Systems:
  Methods of Analysis
  Problem Identification and Solution

The second year Nucleus is an Integrative Field Study Project that is of two quarter duration. Teams of students are placed in a consultant-client relationship with organizations to work on strategic management problems. The subject matter of the Field Study is not necessarily related to the students' Concentration. Many projects do deal with computer systems or information systems and they give additional experience to the student whose major interest is CIS.

All students must demonstrate proficiency in the Common Knowledge Requirements. They embrace the following studies:

Accounting and Finance
Computer Programming (APL)
Managerial Economics
Organizational Behavior
Model Building
Statistics

Some of these are mini-courses that run for only half of a regular Quarter and carry only two units of University credit.

The Concentration in Computers and Information Systems includes required and elective courses from among the following:

Computer Data Processing
Computer Programming Methods
Computer Simulation
Advanced Computer Simulation
Simulation of Operational Systems
Computer Systems Analysis
Computer-Based Management Information Systems
Data Base Management Systems
Special Topics in Computing
Information Systems
Information Systems for Planning and Control
Measurement in Information Systems
Special Topics in Information Systems

The topic of this paper—data base education—must be viewed in the context of the Concentration in CIS and the two year PMP program as they are briefly described in this introduction.

WHY DATA BASE EDUCATION?

Data base management systems are an increasingly important aspect of commercial data processing activities. Program products with names like ADABAS, DMS, EDMS, GIM, IDS, IDMS, IMS, REALITY, S2000, and TOTAL are being vigorously marketed by their vendors. Business firms and government agencies are committing to three to five year DBMS projects with price tags often running into the millions of dollars. It appears that every respectable business with any commitment to the computer in its information processing activity is at least raising the question: "Should we go data base?" This environment is the professional constituency of the management school, especially that portion of the school that directs its attention to computer education. It now appears that every respectable management school with any commitment to the computer in its program of studies may also have to raise a similar question: "Should we provide data base education?" This paper takes the position that the answer to the latter question is Yes. It withholds judgment about the answer to the first question.

Computer science departments regularly teach about data structures. They even infer important uses of these structures for "data bases" and for "management information systems." Doubly-linked lists and rings and inverted lists are a part of the vocabulary of computer science students. Management students may or may not get instruction about data structures. They may only be exposed to the sequential structures of the standard file processing approaches to business data processing that still so dominate. Yet, the commercial world for which they are preparing is moving in the direction of using these structures via data base management systems in implementing computer-based information systems. Both groups, but especially the management students, need some education geared to the practical world of business where DBMS are already in place. A new era in data processing is emerging. It implies a new way to look at the role and use of the computer in business. It places a new demand upon the curriculum of management schools with any program now relating to computer education or with any program now being planned.

THE DBMS COURSE OUTLINE

The data base course that is part of the CIS Concentration has evolved from a Special Topics course first offered in 1967. Different offerings over the years have emphasized readings in the literature, student interviews with vendors about their systems, interviews with users about their experiences, etc. The DBMS course was institutionalized to regular course status in about 1971. The major topics of the present course and the order in which they are presented appear in the following list.

1. Contrast the application and the data base approach to data processing.
2. Review data storage organizations with emphasis upon list structures.
3. Develop a further rationale for data bases that use these data structures.
4. Present an overview of DBMS characteristics that:
   (a) leads to a one page Student Fact Sheet with which to summarize any system;
   (b) summarizes different existing systems using the Fact Sheet;
   (c) underlies assignments to review new systems.
5. Review the CODASYL DBTG Report of April, 1971, and subsequent extensions and recommendations for:
   (a) its relationship to systems studied in (4);
   (b) the record and set architecture recommendations.
6. Design a student-sized data base with:
   (a) one application example worked out for use by all students; and
   (b) other examples from which students may select to develop their own data base.
7. Implementation of the student data base including:
   (a) defining the data base in the DDL of one system;
   (b) preparing test data;
   (c) writing application programs using the DML or a query language available in the system;
   (d) loading a test data base on a computer and executing application programs or queries utilizing the test data whenever and wherever access to the DBMS can be obtained.
8. Review the role of the Data Base Administrator.
9. Review the role of the Data Dictionary/Directory.

The course is covered in one ten-week quarter. Suffice it to say that it is one Quarter of hard work. In its most recent offering in the Winter Quarter, 1975, twenty four students enrolled, with backgrounds ranging from those whose only computer experiences had been within the University to professional system analysts and programmers with as much as a decade of experience.

A STUDENT DATA BASE

The design of a student-sized data base (part six of the course outline) is a critical part of the learning experience. A project that includes design and implementation is the heart of the course. It is based upon the author's longstanding convictions that (1) computer education in management schools must involve the computer; and (2) that a project in which the student does something, however small, is a more valuable learning experience than reading about what someone else has done. This is especially true in the study of data bases where the initial question, "How do we begin?" is a major hurdle in DBMS development.

The data base design starts with a short scenario about an application area. Management questions that demand data from the data base for management decision-making are posed. These yield the relationships and accesses that are necessary to define an initial data base. This initial design is then subject to a number of revisions, some of which are necessary merely to keep the data base in bounds. An example is the purchasing function of a business firm. It is used as a model for students to follow in the data base course while they develop their own data base from a similar scenario and questions in another application area.

The purchasing department of a firm is staffed with BUYERS who authorize PURCHASE ORDERS for the purchase of a single COMMODITY from a SUPPLIER. The supplier fills the order and sends an INVOICE to the purchaser for payment by a certain PAYMENT DATE.

This scenario is obviously an oversimplification of the real business world. It is complicated enough to make an interesting data base and small enough to be wieldy. Management may ask a number of questions during the decision-making processes that arise in the purchasing function. A few simple examples are:

1. What purchase orders has a buyer authorized?
2. What purchase orders have been placed with a supplier?
3. What invoices are due for payment to a supplier?
4. What is the invoice status of a given purchase order?
5. What invoices are due to be paid on a particular date?
6. What total invoice amount is due to be paid by the end of the month?

These six questions are a very small subset of the very large number of queries that can be posed by management. They are sufficient to develop a student data base.

Each question implies both an entry point into the data base and a relationship between two entities in the data base. Other questions would imply relationships among more than just two entities. These would be dealt with in subsequent revisions. The accesses and relationships that stem from just the six questions are listed by question number.

1. Enter at buyer and relate purchase orders to that buyer.
2. Enter at a supplier and relate purchase orders to that supplier.
3. Enter at a supplier and relate invoices to that supplier.
4. Enter at a purchase order and relate invoices to that purchase order.
5. Enter at a date and relate invoices to that date for payment.
6. Enter at the end-of-month date and relate invoices to all prior dates from today's date.

A data base diagram of the different entities, data base entry points and relationships implied by the short scenario and the six questions is shown in Figure 1. Here is a first approximation to a data base design that derives from management data needs to answer management questions.

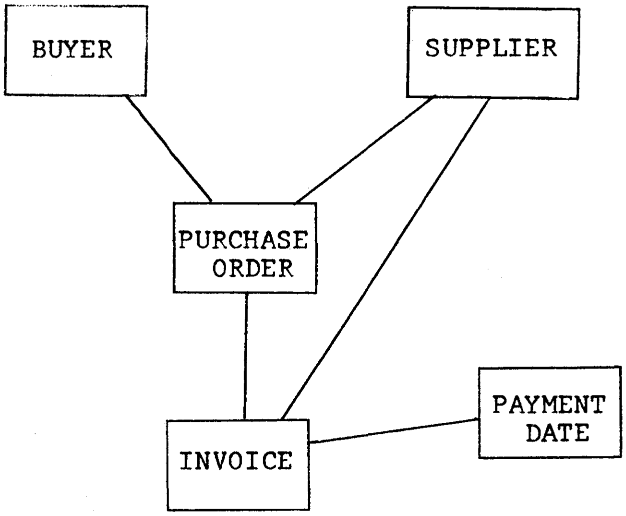


Figure 1—Purchasing data base records and relationships

Figure 1 implies different record types for Buyer, Supplier, Purchase Order, Invoice and Payment Date. These must be fleshed out with data elements that are representative of management needs. Students are already used to defining them as a part of their other computer studies. After a number of iterations of the data base "design" a final decision is made that freezes it for the "implementation."

The next step is to select a DBMS and define the data base in its DDL. It is at this step that students may go in different directions, although as a practical matter, one might select one system as the "course system." The CODASYL DBTG Report and a CODASYL or CODASYL-like system is a place to start. Whether one agrees with its recommendations or with vendors who take a different view, the DBTG recommendations are being widely followed. The Report is having an impact and management students must be aware of this impact. If one follows the DBTG specifications, this means defining the Schema: the Record types and the Sets. The data base diagrammed in Figure 1 has five different record types and (just incidentally) five different sets.

The purchasing data base definition illustrates Owner and Member record types of DBTG. It has one Owner record type (Supplier) that is the owner of more than one set. It has two Member record types (Invoice and Purchase Order) that are a member in more than one set. It has one record type (Purchase Order) that is a member in one set and an owner in another set. The data base does not illustrate multiple record types in a set but this relationship is unnecessarily complicated in an initial student project.

The owner-member relationships expressed by the set derive from the initial design that came from the management questions. Students still have to deal with the set order, that is, whether the member record occurrences are to be stored in a sorted sequence or whether new records will be added to the set in a time-dependent sequence like first-in/first-out or last-in/first-out. They must also deal with duplicate records and whether they will be permitted. They must deal with the possibility that member records should contain owner pointers and the tradeoff between the additional additional storage allocated to the pointer and the computer time to follow a chain of member records to find the owner. Although not a very practical consideration in a student data base, the question is nonetheless important. A number of CODASYL or CODASYL-like implementations are already being marketed. The data base description defined as records and sets is easily translated into the specific DDL of one of them. The exact forms of the DDL are spelled out in vendor manuals and these are usually readily obtainable.

The next step is to define the data forms for the different record types and prepare test data. This in itself is a valuable learning exercise because the data must match the different entities in the data base. Buyer records must contain buyer keys and supplier records supplier keys that are known and will be used in the purchase order and the invoice. A purchase order must have valid buyer and supplier keys. An invoice must reference a supplier that is defined in the data base and the commodity and purchase order that are also defined. The payment date in an invoice must be a date that is in the calendar implied by the payment date record. Devising test data is a non-trivial task in the data base environment.

Finally, the data must be loaded into the data base. Some systems provide load utility programs. For others, one must write load programs in, say, COBOL to enter data into the data base. Application programs can then be written using the DML or a Query language to process the data in the data base. This step obviously implies access to a computer.

One can stop with a CODASYL-like system. A richer educational experience is provided when different students can work with different systems. The purchasing data base has been defined in a number of different DBMS and is available to students as a model for them to follow. Two DBMS that provide striking contrasts with the DBTG architecture are ADABAS and S2000.

ADABAS is a partially inverted file system and Figure 1 implies only four different record types in it because the payment date in the invoice can be inverted to establish the access and relationship to invoices. The ADABAS notation is completely different from that of COBOL that is the basis of the DDL for most CODASYL-like systems. It provides a loader for the initial test data. It has a limited query language, ADASCRIPT. It contrasts also with the DBTG Schema by making many of the relationships in the application program or query through the inverted lists rather than in the data base description.

The S2000 data base system forces one to revise the data base model shown in Figure 1 to accomplish the same accesses and relationships. The data base is defined in a single hierarchical format. One must decide what is to be at the top of the hierarchy. Supplier seems to be the logical choice for this place in the purchasing data base with purchase orders repeating for each supplier and invoices repeating for each purchase order. Since S2000 is also an inverted system, access is established by defining certain data elements as keys. This includes the payment date in the invoice. The single hierarchical structure of S2000 places the buyer data in each purchase order so that, whereas it is separate in both prior systems, it is now a redundant item in each purchase order record.

The point is that contrasting systems provide a richer educational experience. One can pick and choose from among a dozen systems to make the contrasts or avoid them as one sees fit.

A critical assumption underlying the student project is that an operating system is available. Without access to a system, the project stops short of an implementation. An implementation on a computer is desirable but not necessary. The project can end with a description in the DDL, with test data and with simulated queries or application programs. This is a valuable learning experience that suffers only because one is never quite certain that the data base is defined correctly, that the data will load and can be processed correctly. A project could stop short of even the data base description in the DDL of an existing system and still provide students with substantial learning. In this case, the DBTG Schema is probably the wisest choice because it emphasizes an approach that many vendors are following.

In short, one may "implement" Part 6 of the course outline in a number of ways. Design of a simple data base from a short scenario and a list of questions is a must. Description in the DDL of some system adds a real-world flavor to the project. Loading the test data base and running application programs or queries against it is the ultimate test that the whole data base really hangs together. The satisfaction that derives from this ultimate test is not present when one aborts the project at the design or descriptive stage.

REACTION TO THE DATA BASE COURSE

Student reaction to the data base course in its present form is consistently that it is a lot of work. The general impression is also that the work is worthwhile because it provides an in-depth understanding of data base development although on a small scale. This is in addition to rather broad knowledge about DBMS characteristics and the descriptive materials that abound. Computer science students, even in prior offerings have been known to say that for the first time they understood the importance of the different structures they had studied and why they had studied them in the first place. During the Winter Quarter, 1975, some students were initially turned off because the course materials presented to them were the next-to-final draft of chapters in a forthcoming book. Other students seemed to delight in finding errors in the same chapters. Student projects defined five different application data bases in the DDL of nine different DBMS.

The most recent "community" reaction is perhaps best summarized by the willingness of a number of different companies to provide access to a DBMS on their computer. Implementations that included loading data into the data base and accessing either with an application program or a query language included the UNIVAC DMS 1100, ADABAS, S2000 and Xerox EDMS. Both vendors and users cooperated in this venture and programs have been run in both batch and terminal based systems. The support can be interpreted not only as support for this course but also for the general need for DBMS education in management schools.

SUMMARY

This paper takes the position that data base instruction is a necessary part of management education within the framework of broader computer and information system studies in management schools. It has presented the outline of a specific course that offers not only descriptive coverage of DBMS but insists also upon a laboratory project in which students design, describe, and implement one small data base in an existing system. Community support to provide access to systems so that students may load a data base and process it is both welcome and heartening. Perhaps this approach and some of its details will be useful to others who are thinking about data base instruction and how it should be a part of "Computer Education for a Computerized Age."

Computers in architectural education

by JENS G. POHL

California Polytechnic State University
San Luis Obispo, California

THE "ART" OF BUILDING DESIGN

It is perhaps difficult to understand that even today the design of the built environment is best described as an "art" rather than a "science." The designer or architect is an artist who relies heavily on creativity and intuition to solve technological problems.

The individual components, such as structure, cladding, mechanical and electrical services, furnishings, fittings and finishes, which are available for the design and construction of buildings have all been developed and tested in a scientific manner and may therefore be classified under the heading of technology. However, the occupants of buildings are people whose behavior, needs and expectations are governed by innumerable social and cultural interactions, many of which are not easily defined. The establishment and analysis of the interrelationships between the large number of technological components and an even larger number of behavioral patterns is the formidable task of the architect. Faced by the apparently insurmountable problem of analyzing (and perhaps optimizing) an almost incomprehensible number of largely undefined variables, it is no wonder that the architect has in the past and continues today to use intuition (or shall we call it creativity) to overcome the shortcomings of existing scientific methods. The fact is, that even today the range of operation research techniques available to the designer is entirely inadequate for solving, let alone optimizing, a complex building design system.

Perhaps the design systems could be simplified? In the past, this appealing approach has led to some horrendous failures. Simplification may be achieved by reducing the number of variables in the problem system. However, this cannot be accomplished successfully without full knowledge of the relative importance of not only the variables themselves, but also the interactions between the variables. Unfortunately, the state-of-the-art in the social and behavioral sciences cannot provide the architect with a comprehensive set of defined variables, ranked or unranked.

The alternative approach, which consists of subdividing the design system into a large number of often complex subsystems (e.g., structure, environmental control, user functions, etc.) to be solved more or less in isolation, has been more successful. For example, this piecemeal approach would allow the natural and artificial lighting design of a building to be completed independently of the design of the structural support system. There remains, however, the problem of determining the nature of the interface between each of the subsystem solutions.

It is this second approach which is in common use in architectural practice today. Scientifically based procedures have been developed for solving many of the subsystem problems which make up the building design system. All of these procedures are readily computerized into interactive and non-interactive software packages. Unfortunately, the integration of these separate solutions into one optimized design solution cannot be computerized prior to the development of considerably more sophisticated interface procedures. Existing operation research techniques appear to be entirely inadequate for the optimization of large building design systems.

COMPUTERS IN ARCHITECTURAL PRACTICE TODAY

The adoption of even the most elementary computer applications in architectural practice has been slow. Faced with the overwhelming complexity of the design system, the architect has tended to assume the role of a coordinator responsible for the integration of the various subsystem solutions mostly prepared by specialist consultants, such as structural engineers, electrical and mechanical engineers, building economists and construction engineers. This is particularly true for the design of large commercial buildings, where the final integrated design is very much a compromise solution based on fact, presumption and a considerable amount of intuition. The inability to rely on established numerical methods for the final and most important stage of the design process has made most architects inordinately suspicious of all mathematically based design procedures. It is therefore only natural that the same suspicion should have carried over to the "computer," which has in the past been represented to the architect as a high speed calculator rather than a powerful data-processing and simulation tool.

To make matters worse, a number of early computer users in the contracting field experienced critical setbacks in their efforts to implement computer-based operations. At times, this has led to the complete rejection of computers as timesaving tools by individual contracting com-

TABLE I—Typical computer applications in the building industry

TYPE OF PROFESSIONAL SERVICE	AVAILABILITY OF COMPUTER PROGRAMS	MOST APPROPRIATE COMPUTER FACILITY
Regional survey:	Abundance of programs dealing with the anlysis of census and survey information.	Batch-processing
Land use planning:	Inventory programs capable of classifying land characteristics, mostly using overlay techniques.	Batch-processing with access to plotter.
Transportation planning:	Large scale mathematical simulation models which enable user to experience visual movement through designed space, also overlay and support facilities analysis and design programs.	Time-sharing with access to graphic display units for simulation programs. Batch-processing with access to plotter for inventory and overlay programs.
Infra-structure planning:	Mostly design programs dealing with district heating, cooling, water supply and waste disposal. Programs dealing with the integration of services are as yet scarce.	Time-sharing with access to graphic display unit desirable (Batch-processing with access to plotter for some applications).
Building development feasibility study:	Programs normally based on local building regulations and specific building types, such as residential and commercial.	Time-sharing
Project time scheduling:	Numerous programs available to analyze and update PERT networks.	Time-sharing (Batch-processing for large projects).
Site analysis:	Contour analysis and cut and fill calculations	Batch-processing with access to plotter
Design brief development:	Report generating programs with the ability to establish correlations, contradictions, and omissions.	Batch-processing.
Establishment of activity and space relationships:	Programs capable of forming activity and space interaction matrices	Batch-processing or time-sharing if input and output not large

TABLE I—(continued)

Comparative space groupings:	Two and three-dimensional layout optimization programs.	Time-sharing (Batch-processing if user interaction not required).
Comparative heating-cooling, ventilation, lighting, acoustic and structural analyses:	Increased availability of programs which establish environmental and structural design guidelines.	Time-sharing.
Preliminary cost analysis:	Programs in which the accuracy achieved is related to the degree of detail of the input.	Time-sharing.
Preparation of design drawings:	Small number of recently developed programs allow graphic display unit to be used as sketch-pad. Programs are often linked to data bases incorporating fittings and standard drafting symbols. Abundance of perspective programs with and without ability to delete hidden lines.	Time-sharing with access to graphic display unit for design process. Batch-processing with access to plotter for presentation drawings.
Structural design:	Abundance of comprehensive and specific programs mostly directed toward structural analysis rather than design.	(Batch-processing for very large programs). Time-sharing.
Environmental design (acoustics, lighting, heating-cooling and ventilation):	Mostly batch-processing programs, although the number of interactive time-sharing programs is rapidly increasing.	Time-sharing.
Design and integration of mechanical and electrical services:	Many design programs, but very few deal with the relationships between two or more support services.	Time-sharing (Batch-processing with access to plotter for layout drawings).
Detailed cost estimate based on design drawings:	Programs mostly owned by large design offices, construction firms and cost consultants.	Time-sharing.
Preparation of specifications:	Complete specification service provided by specialist firms.	Batch-processing with access to high speed printer.

TABLE I—(continued)

Preparation of working drawings:	Few programs available to date, due to the requirement of comprehensive data bases incorporating materials, fittings, doors, windows and standard construction details.	Time-sharing (preferably with access to digitizer to transfer dimensions and quantities directly from drawings to computer).
Development of bidding strategies:	Small number of interactive construction management games, which enable the user to develop bidding and scheduling strategies on the basis of simulated conditions.	Time-sharing.
Project scheduling:	Large number of CPM and precedence diagram programs available. These programs are capable of generating calendar time schedules and allocating manpower and equipment resources.	Batch-processing (time-sharing may be economical for small construction projects).
Equipment records and costs:	Resource leveling and allocation programs require large computers and are normally based on CPM network analysis.	Batch-processing.
Accounting, payroll and record keeping:	Many programs available, which are based on time card input and prepare payroll checks, financial statement and productivity reports.	Batch-processing.
Development of progress cost reports:	Programs normally record direct and indirect costs in various categories and prepare progress claims at monthly intervals during the construction phase of any size building project.	Time-sharing and batch-processing.
Interior planning and space utilization reports:	Typical programs allow building plans to be transferred from drawings to computer, to form the basis of new layout designs and their comparative evaluation (e.g., department stores).	Batch-processing in conjunction with digitizer to transfer existing layout from drawing to computer.

panies. In most cases these teething problems can be traced to the existence of poor communication channels between the construction expert and the computer specialist. Typically, the computer specialist would embark upon the development of a program, blissfully unaware of the fundamental characteristics of the building industry. His only source of information would have been a construction expert who neither appreciated the type of information required for the production of a meaningful program, nor had the ability to foresee the interpretation likely to be placed on his information by the computer specialist. Inevitably, the end result was a computer program which did not save time and yet required the contractor to adopt a most costly office procedure. These unfortunate circumstances forced some early pioneers of computer applications in the building industry to become staunch supporters of traditional practices.

Fortunately, today the question is no longer whether architects should make use of computers, but rather, how can they best capitalize on existing computer capabilities?

The typical computer applications listed in Table I are all related to the solution of subsystems within the overall building design systems. Applying computers to isolated tasks, in this manner, is not necessarily cost effective. The real benefits are obtained when these separate tasks are linked together by a common data base. Unfortunately, many obstacles stand in the way of the development of central data bases in the architectural profession. The flow of information in the building industry is complicated by the relatively large number of decision makers involved. The architect is only one principal in a large open system incorporating the building owner, structural engineer, material supplier, fabricator, mechanical and electrical engineer, landscape architect, contractor and, in most countries outside the U.S., the quantity surveyor or building economist. Each of these decision makers speaks a slightly different language requiring a certain amount of translation.[1] Simplification of the open system shown in Figure 1 by the addition of a central computerized data base (Figure 2), presupposes the existence of a high degree of standardization. It is only during very recent years that trends have become apparent, which suggest that the building industry is at last undergoing the painfully slow process of preparing and adopting uniform standards, codes, regulations, and documentation and information classification systems.

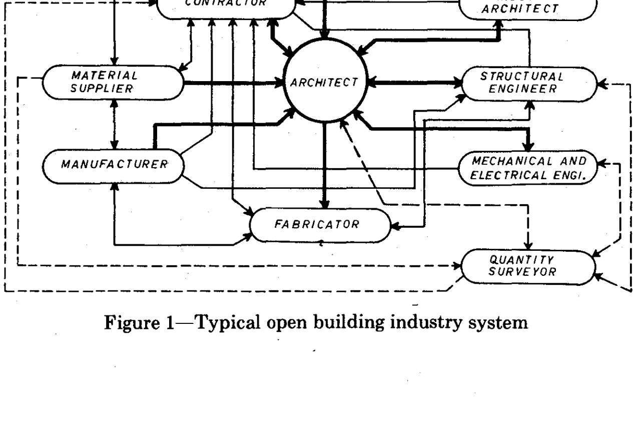


Figure 1—Typical open building industry system

ARCHITECT—COMPUTER COMMUNICATION

Architects communicate most efficiently with scale drawings and sketches, and therefore require access to computer systems incorporating graphic devices, such as plotters, digitizers, light pens and cathode ray tubes. In addition, a high degree of interaction is desirable for most architectural computer applications. While interactive time-sharing systems are now widely available to the building industry, computer graphics systems are thinly spread and relatively expensive. As far as the educational sector is concerned, few schools of architecture have access to even the most elementary graphics facilities. In the minority of cases where a university can claim a graphics capability, more than likely the utilization of this capability is severely limited by the absence of adequate interface facilities between the graphics system and the other batch-processing and interactive time-sharing computer facilities available on campus.

Perhaps the architect's need for "interactive" computer systems requires further explanation. The design process involves long decision chains coupled with the progressive evaluation of information sets, in which each decision stage may affect both the type of information to be considered and the evaluation procedure. In this situation, there is a critical need for direct interaction between architect and machine. Firstly, the architect cannot afford to pause in the decision sequence while cards are punched and a program is processed in batch mode. Secondly, the architect must be able to change the sequence of the decision chain without modification of the computer program. Ideally, the computer might become a partner in the design process, with the ability to respond intelligently to the proposals made by the architect. In the absence of computers with some degree of artificial intelligence, the interactive time-sharing system appears to provide the best vehicle for satisfying the rapid response requirement.

SHOULD THE ARCHITECT "PROGRAM" THE MACHINE?

Architectural educators have been asking themselves this question for some years now. Their task would be made so much easier if there existed a set of standard computer programs used on a day-to-day basis by the building industry. Unfortunately, the very existence of such standard programs would severely limit the application potential of computers. Developments in design methods over the past few years have demonstrated the strong influence of computers in shaping the methodologies of the 1980's. At the same time, present and

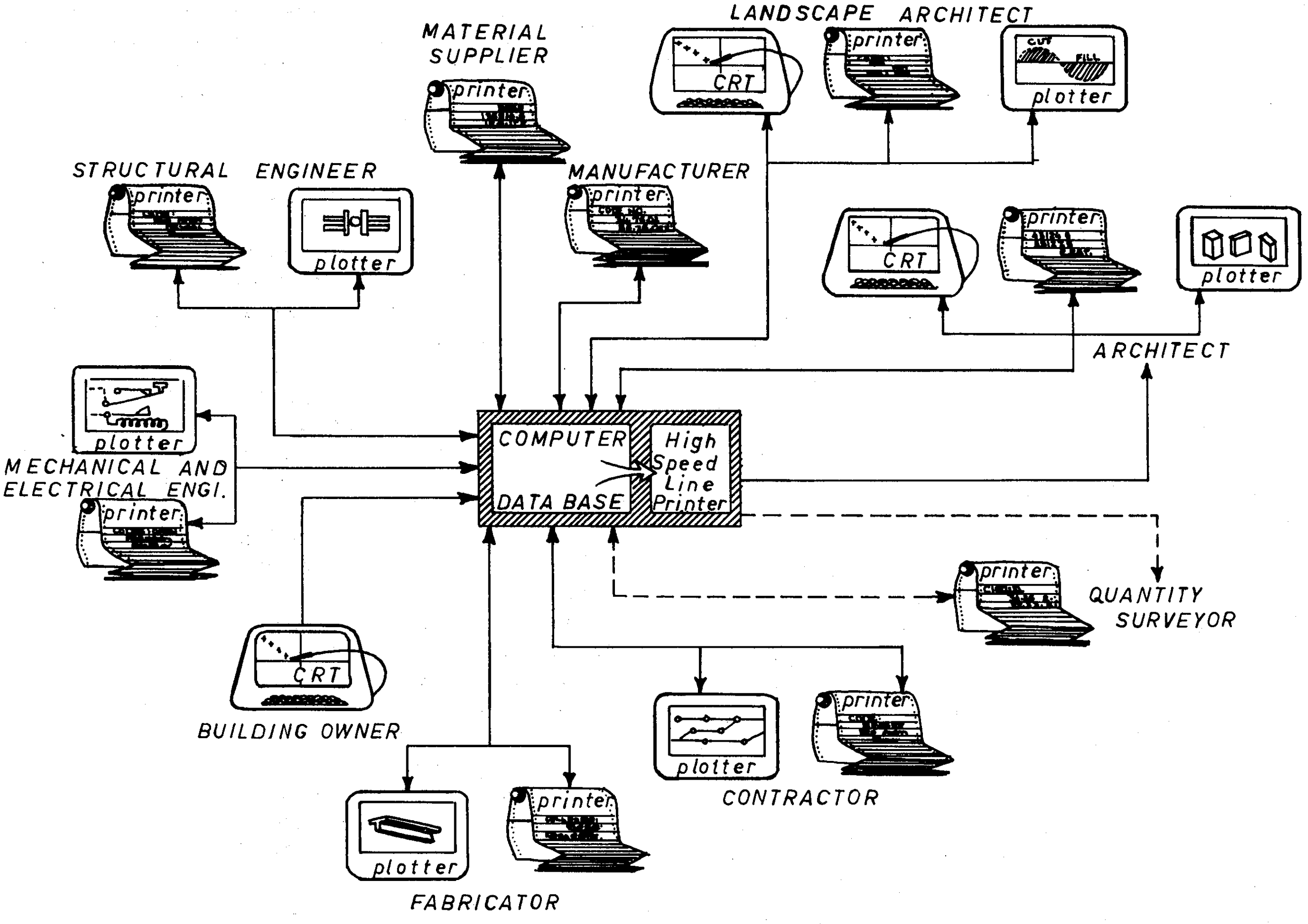


Figure 2—A computerized information system

future developments in computer technology are likely to provide the building industry with tools so powerful that they will have a profound influence on the scope and effectiveness of the services offered by the practicing professional. It is essential that the architect should have direct input in this development process and actively participate in the formulation and coding of programs, which will embody the most suitable planning, design and construction methodologies. The prospective building owner of the 1980's is likely to expect not one or two but eight to ten alternative design solutions for a major building complex, each alternative to be based on a detailed analysis of a large volume of accurate information. To achieve this standard of service, the architect must have access to comprehensive data bases and fast retrieval systems. There is no doubt that the computer will constitute the most inexpensive means of providing both.

It would be a grave error to look upon the computer as no more than a high-speed desk calculator or super slide rule. One of the most inspiring capabilities of the computer is its ability to "simulate" entire systems and perform long sequences of operations. The effect of a single computer program in influencing the end product will therefore be much greater than has been the case with the application of less powerful design and planning tools in the past. To ensure that the control of the various planning and design processes continues to be exercised by the designer, the latter will need to be actively involved in the development of computer programs. In short, the practicing architect will require not only a general knowledge of computer systems, capabilities and limitations, but also a working knowledge of at least one computer language. Disregard of this requirement will eventually lead to a serious loss of influence and credibility of the architect. Unfortunately, some minor symptoms of a lack of professional foresight and responsibility in this area are already today a fact of life. There is a sizable number of computer programs in use today, particularly in educational institutions, which are of unknown origin, which have been subjected to virtually no testing and which are based on unexplained and sometimes erroneous theories. To make matters worse, the untrained user tends to be extremely gullible and will often allow the most incredible computer results to override sound judgment based on experience and common sense.

WHAT SHOULD BE TAUGHT?

The need to train undergraduate architecture students in the computer field was first recognized by schools of architecture in the middle 1960's. In those early years, most architectural educators with little or no knowledge of computers found themselves at a complete loss to develop and implement in-house computer courses. Under these circumstances, it was found to be most convenient to place this responsibility into the eager hands of mathematics, science and computer science departments. In fact, it was most common at that time as it is still today in some schools of architecture, for computer course material to be considered an adjunct to if not synonymous with mathematical topics, such as calculus.

The Boston Computer Conference of 1964, the first formal gathering of computer enthusiasts in architecture, was attended primarily by a group of young architectural educators from the greater Boston-Cambridge university complex.[2] These pioneers were looked upon by their senior colleagues, and indeed by the profession at large, as misguided theoreticians with a warped view of architecture. Quickly the term "computer freak" was coined in academic circles, as a form of resistance to the possible insurgence of this new electronic monster into the inner precinct of architecture, namely the design process. Naturally, this atmosphere was not conducive to the systematic evaluation of potential computer applications in schools of architecture, nor did it lead to the careful structuring of effective computer courses.

A decade later, with most of the early paranoia dissipated, it is high time that architectural educators should re-examine the structure and content of existing computer courses. Which of the following instruction levels[3] is most appropriate for architecture students?

(1) A general descriptive course dealing with a variety of typical computer applications.
(2) A descriptive course which deals not only with typical computer applications, but also provides the architecture student with a working knowledge of a number of existing computer programs.
(3) A description-implementation course which also incorporates elementary computer programming.
(4) A complete computer programming course incorporating also detailed application information.
(5) A rigorous computer science course.

In a general descriptive course (i.e., course (1)) the student learns little or nothing about computer systems, but gains only a general knowledge of existing computer applications in the profession. This type of course tends to be ineffective and unconvincing, while providing the student with little information and no experience for hands-on operation at a later date. Although the student comes into direct contact with computers in course (2), he is forced to use programs which he does not fully understand. This is a very dangerous situation and could easily lead to the acceptance of computer results without adequate control over the accuracy of the program. Course (3) allows the student to develop simple programs and understand existing programs. Not only does this type of course emphasize the importance of checking programs for logic and syntex, but it also forces the student to define the problem in a definitive manner. This course meets most closely the special requirements of architecture students.

Course (4) is more suitable as a follow-up elective course, for the interested architecture student who wishes to gain more in-depth computer knowledge for possible specialization after graduation. A rigorous computer science course (i.e., course (5)) is not suitable for architecture students at any undergraduate level, since it would unnecessarily make programmers of the students.

WHO SHOULD TEACH?

Many computer applications in architecture are so strongly interwoven in the design process, that the computer is not just a tool, but an integral part of the design methodology. In this respect, computer applications often form the very basis of the development of new design methods. From this point of view alone, it is most important that the computer course should be offered inhouse rather than by an outside school or department. Moreover, computer service courses are normally offered to students in a wide range of majors, with the result that programming examples tend to have little direct relevance to any particular major. Experience has shown that architecture students, in particular, find it difficult to separate the programming aspects in advanced mathematical examples, since they are unfamiliar with both the mathematics and the computer programming. Finally, there is the important consideration of faculty participation in an in-house computer course. Not only does the inhouse course provide an opportunity for faculty training, but it also encourages the integration of computer applications in concurrent and subsequent non-computer courses.

A CASE STUDY

The computer course presently offered in the School of Architecture and Environmental Design, California Polytechnic State University, San Luis Obispo, has been developed to meet the specialized requirements of undergraduate students in architecture and related majors. The School of Architecture and Environmental Design is the largest school of its kind in the United States with a total enrollment of around 1500 students. While the school offers undergraduate degrees in architecture, architectural engineering, city and regional planning, landscape archi-

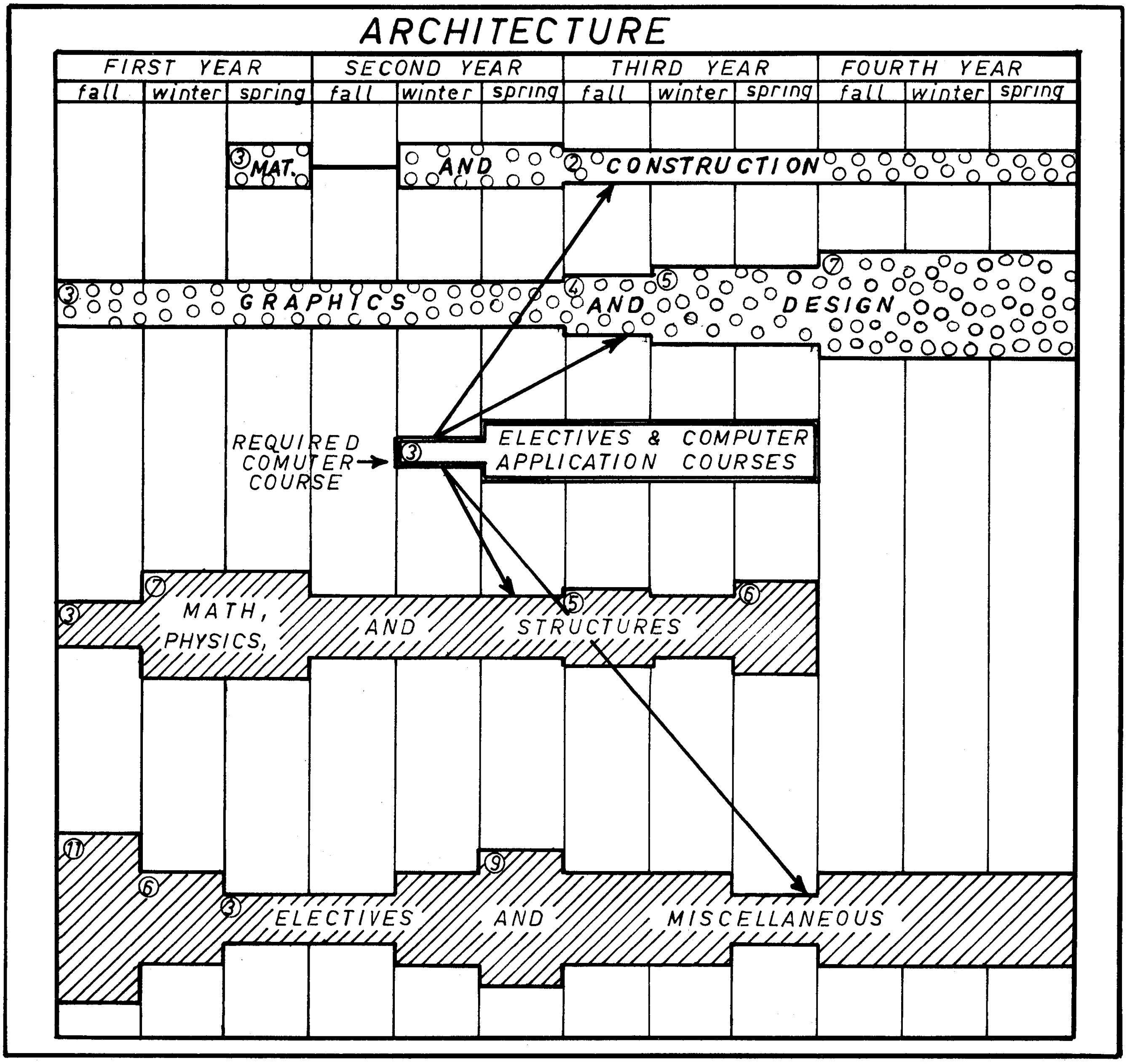


Figure 3—Computer course structure in the School of Architecture and Environmental Design of the California Polytechnic State University, San Luis Obispo

tecture and construction engineering, the architecture program carries by far the largest portion of the enrollment. All students are required to take a two credit unit computer course during their sophomore year.

Approximately 120 students are enrolled in the computer course each quarter. To avoid unnecessary duplication and yet retain a strong tutorial emphasis, the course is divided into a one hour lecture and a three hour tutorial sequence each week. The lecture is attended by all students concurrently, with the subject matter shared among participating faculty for presentation. In this manner, each instructor is encouraged to develop his or her interest area into a comprehensive lecture series.

The tutorial sessions are attended by approximately 24 students and are intended to extend the lecture material, as well as provide assignment assistance on a group and individual basis. The course is designed to introduce students with no previous knowledge of computers and computer applications in architecture to the capabilities and limitations of computer systems and their utilization in environmental design, with the following specific objectives:

- To provide students with a general understanding of computer hardware and the peripheral units which are normally combined in batch-processing, interactive time-sharing, remote job entry and graphical computer systems.
- To explore the existing capabilities and limitations of computers in relation to existing and potential architectural applications.
- To familiarize students with existing computer applications in architecture and, in particular, the range of computer programs available to students in the school.
- To acquaint students with the esoteric vocabulary which has been developed as a byproduct of the growth of computer applications in all fields.
- To provide students with a working knowledge of one computer language (e.g., FORTRAN IV, BASIC, etc.), to enable them to code relatively simple computer programs and understand listings of existing programs.
- To introduce students to procedures for defining problems and structuring information prior to computerization.
- To familiarize students with computer program documentation and checking procedures.

Emphasis is placed on the development of user experience by the student and the integration of this newly gained knowledge in concurrent and subsequent courses.

While it is not intended for the architecture student to become a computer expert, follow-up elective courses are offered both within and outside the school for those students who wish to extend their knowledge in this field (Figure 3).

Course content

Computer languages, by and large, have been developed for scientists to facilitate the solution of mathematical problems involving complex and repetitious calculations requiring little input and output capability. Most architectural problems, on the other hand, require only a modest degree of computing power, but depend on the transfer of large quantities of data (i.e., numeric and alphameric) to and from the machine. In addition, the architect has in the past and will continue to rely largely on graphical means of communication. To date no computer language specifically designed for the solution of architectural problems has been developed.

If we follow the commonly accepted classification of computer languages into machine-oriented, procedure-oriented and problem-oriented, then it is without doubt the procedure-oriented language which most closely fulfills the solution needs of an architectural problem. Of the half-dozen or so major procedure-oriented languages available today, FORTRAN IV was selected to form the basis of the course. The principal reason for this choice being the availability of FORTRAN on almost every substantial computer system, today. From time to time, BASIC has come under serious consideration, not because of its simplicity, but because it is often the only language available on low cost mini-computer and programmable calculator systems.

The course, as it is taught now, incorporates ten lectures structured in the following manner:

FIRST WEEK: *Introduction*: What are computers and how do they work? Computer languages and computer systems.

SECOND WEEK: *Fortran IV*: Statement types; integer and real constants and variables; operators; functions and expressions; ARITHMETIC statements; STOP and END statements; PRINT statement (Watfor compiler).

THIRD WEEK: *Input and Output Statements*: Example No. 1 batch-processing program (Watfor compiler); READ, WRITE and FORMAT statements; time-sharing terminals as input and output units; Example No. 2 time-sharing program.

FOURTH WEEK: *Transfer Statements*: Unconditional GO TO statements; conditional IF statements; COMMENT statement (time-sharing and batch-processing); Example No. 3 time-sharing program.

FIFTH WEEK: *Graphics*: Computer graphics in architecture; large graphics systems, minicomputers and programmable calculators.

SIXTH WEEK: *Graphics (Cont.)*: Hewlett-Packard 9830 graphics system. *Site Planning*: Computer application in site analysis and mapping; application of the GRID (batch-processing) program.

SEVENTH WEEK: *Site Planning (Cont.)*: Application of the GRID (batch-processing) program.

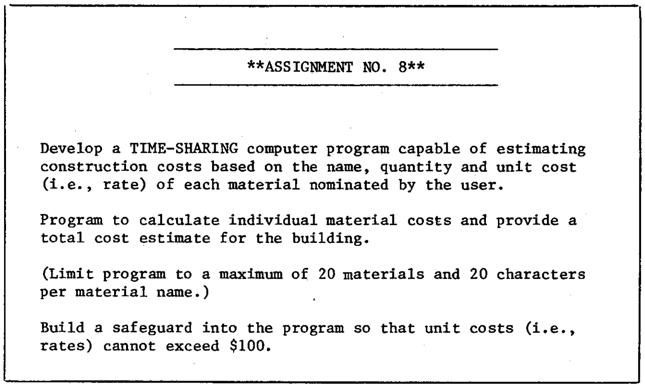
EIGHTH WEEK: *Repetition*: The DO statement—acting as a counter—acting as a number generator—controlled by program user; Example No. 4 time-sharing program.

NINTH WEEK: *Arrays*: What are arrays? Comparison of arrays and simple variables; DIMENSION statements, arrays in input and output statements; Example No. 5 time-sharing program.

TENTH WEEK: *Data Files*: Data file input statements; data file entry under the EDIT subsystem; data storage and retrieval; Example No. 6 time-sharing program.

These lecture topics are strongly reinforced by tutorials, during which the subject matter of the previous lecture is extended by suitable programming examples, typical executions of existing programs, films, demonstrations of in-house computer facilities and visits to the University Computer Center.

In addition, students are required to submit one assignment each week. All of the assignments require access to the University computer facilities with emphasis on the



ASSIGNMENT NO. 8

Develop a TIME-SHARING computer program capable of estimating construction costs based on the name, quantity and unit cost (i.e., rate) of each material nominated by the user.

Program to calculate individual material costs and provide a total cost estimate for the building.

(Limit program to a maximum of 20 materials and 20 characters per material name.)

Build a safeguard into the program so that unit costs (i.e., rates) cannot exceed $100.

Figure 4—Typical computer programming assignment

interactive time-sharing system. Assignment topics cover a wide range of architectural subjects and include programming exercises (Figure 4), as well as the execution of a variety of existing programs (Figure 5).

Support material

At the time of implementation of the course a review of available publications revealed a singular lack of any comprehensive computer textbook suitable for undergraduate architecture students. Existing texts could generally be divided into three categories.

DESCRIPTIVE texts written for the practicing architect, which describe areas of architectural utilization of computers, but do not address themselves to problem definition and programming.

USER-ORIENTED texts written for computer users in a number of disciplines, such as engineering, social science and psychology, but not architecture.

SPECIALIST computer science texts.

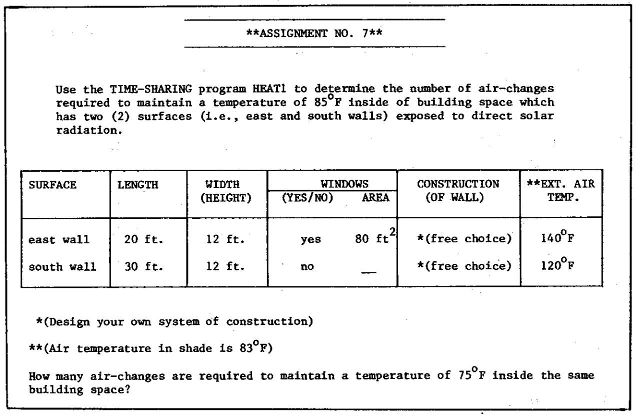
To overcome this problem, a textbook incorporating both a description of computer hardware, computer systems, implementation alternatives and elementary FORTRAN IV programming was prepared and published by faculty instructing in this area.[4]

During the early implementation period of the course, it was recognized that the type of computer program useful to the building designer could not be developed by the architect during the design process. Therefore, to facilitate the integration of computer applications into all phases of the architecture degree program, the school immediately embarked upon the establishment of an architectural program library. With an abundance of batch-processing computer programs commercially available from other educational institutions and industry, the limited need for this type of non-interactive program was soon satisfied.[5] Unfortunately, the same situation did not apply to interactive time-sharing software. Even today, there is a singular lack of commercially available architectural time-sharing programs, and the few programs which are offered for sale by computer service bureaus are well outside the financial means of the entire California State University and Colleges System, let alone an individual school. Faced with this situation the School of Architecture and Environmental Design undertook the time-consuming task of software development. Over a period of two years faculty and senior students were able to develop more than 20 major interactive time-sharing programs, ranging from the design of environmental control systems (i.e., natural and artificial lighting, acoustics and heat transfer) to structural analysis, computer-aided design and construction management games. Each program is carefully documented (i.e., program name, language, author, disclaimer, program description, limitations, operating instructions, input and output, theoretical basis) and, after a period of testing, groups of programs are formalized into printed manuals.[6] These manuals are available to students at production costs through the University bookstore. The school has now obtained approval from the University to market these computer programs commercially, and it is hoped that in the near future sales profits will in themselves sustain the entire program development operation.

Computer facilities

The school presently has access to the following computer systems:

1. BATCH-PROCESSING (LOCAL): IBM 360/40 computer system with 180 K (approx.) core storage, 1100 lines per minute printer, 1000 cards per minute reader, 300 cards per minute punch, three disc drives, controller and tape drive.
2. TIME-SHARING (BATCH OR RJE): IBM 360/20 computer system with 8 K core storage, 315 lines per minute printer, 120 cards per minute reader and 90 cards per minute punch, which serves principally as



ASSIGNMENT NO. 7

Use the TIME-SHARING program HEAT1 to determine the number of air-changes required to maintain a temperature of 85°F inside of building space which has two (2) surfaces (i.e., east and south walls) exposed to direct solar radiation.

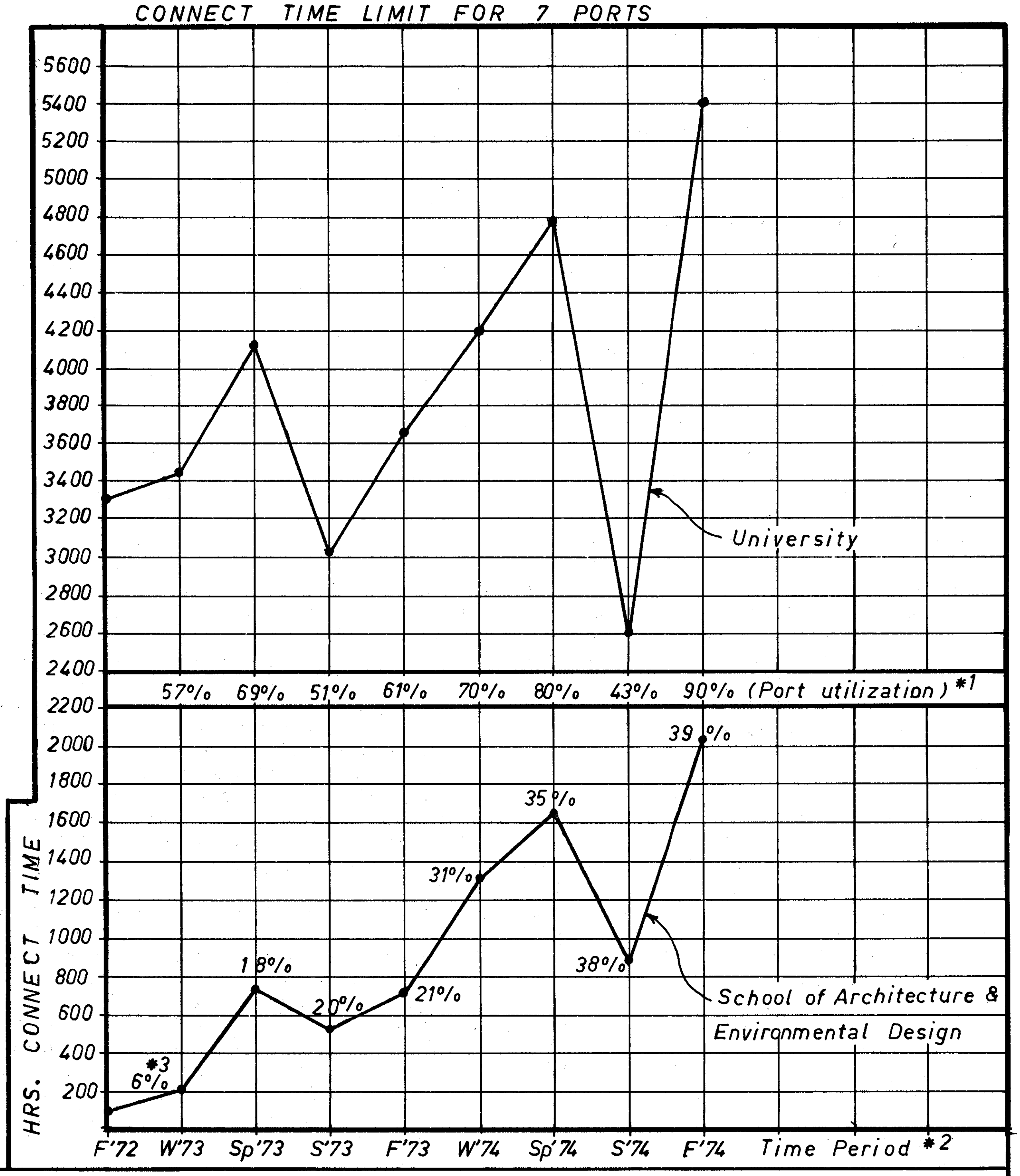
SURFACE	LENGTH	WIDTH (HEIGHT)	WINDOWS		CONSTRUCTION (OF WALL)	**EXT. AIR TEMP.
			(YES/NO)	AREA		
east wall	20 ft.	12 ft.	yes	80 ft^2	*(free choice)	140°F
south wall	30 ft.	12 ft.	no	—	*(free choice)	120°F

*(Design your own system of construction)

**(Air temperature in shade is 83°F)

How many air-changes are required to maintain a temperature of 75°F inside the same building space?

Figure 5—Typical assignment requiring the execution of an existing computer program



*1 7 Ports available an average of 13hrs. per day for 7 days a week.
*2 Each time period is equal to 9 effective instructional weeks.
*3 Percentage of total university time-sharing usage.

Figure 6—Time-sharing computer usage by the School of Architecture and Environmental Design at the California Polytechnic State University, San Luis Obispo

a Remote Job Entry terminal to the State University Data Center CDC 3300 dual processor installation.

3. TIME-SHARING (INTERACTIVE): Central time-sharing network supported by a CDC 3170 dual processor computer installation located at California State University, Northridge. Of 96 ports distributed among the 19 campuses of the California State University and Colleges System, seven (7) ports are available on this campus.
4. GRAPHICS: Until recently the university leased an IBM 2250 graphics terminal connected to the local IBM 360/40 computer system. This has now been replaced by a mini-computer graphics system consisting of a PDP 11/35 processor, two display tubes and a 36 in. CAL-COMP drum plotter. A second CAL-COMP plotter is presently supported by an IBM 1130 computer system at a nearby community college, under a limited time exchange and verbal agreement. In addition, the school owns a Hewlett-Packard 9830A programmable calculator which supports a digitizer, plotter, cassette tape unit and printer.

The utilization of these facilities by the school has increased steadily during the past three years. However, by far the most dramatic increase in usage has been in the interactive time-sharing field, where the school now uses over 40 percent of the entire time-sharing capability available to the University (Figure 6). This extraordinarily high usage level has been achieved despite the saturation of all computer systems on campus and the presence of strong computer science and engineering departments.

PROJECTIONS FOR THE YEAR 1980!

By far the most critical problem facing the continued development of computer applications in the School of Architecture and Environmental Design at the California Polytechnic State University, San Luis Obispo is the complete inadequacy of the existing computer facilities available on and off campus. Despite the frequent breakdown of teletype and CRT terminals, the university normally achieves a time-sharing port utilization in excess of 85 percent. At the same time, job turnaround times on the RJE and local batch-processing systems are in the order of 10 to 15 hours and often reach 30 hours toward the end of each academic quarter.

Conservative estimates based on a detailed analysis of courses offered by the school indicate that by 1980, architecture students will require approximately 1,000 time-sharing connect hours per week (i.e., 40,000 hours per year), with access to at least 58 ports (i.e., based on 17 hours of time-sharing availability per day). These estimates presuppose that the time-sharing system will have a graphics capability, supported by suitable input-output devices at each fifth port.

In addition, by 1980, the availability of large data bases will have become an integral component of architectural computer applications. The full utilization of such data bases will require greatly improved system interfaces, to facilitate the transfer of data from one computer system to another. For example, a user will expect to be able to access a regional land-use data base on an interactive time-sharing system, transfer a sizable parcel of information to a larger computer system and process this information using a data analysis program of the capability of SYMAP, in batch mode.

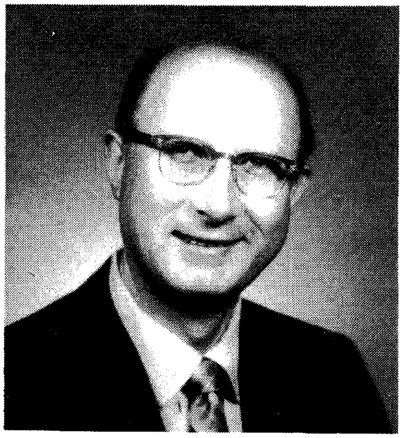
It would appear that university computer center managements, in general, are not fully aware of the implications of an almost explosive increase in computer usage by non-computer science students. There is no doubt that during the next decade computer science departments, which now hold the privileged position of principal computer user on most campuses, will relinquish this position to users of library programs, such as students majoring in social and behavioral science, business administration, agricultural science, engineering, medicine, biological sciences and architecture. This change in user distribution will necessitate university computer centers to broaden the scope of their services by providing increased assistance to non-computer science users. Moreover, some attention will need to be given to the streamlining of job entry procedures. For example, in the California State University and Colleges System students wishing to access library programs under the RJE batch system are normally required to submit at least five control cards with each job (e.g., the SYMAP program may require up to 20 control cards). Needless to say, the architecture student learns absolutely nothing about environmental design by struggling with the interpretation, punching, assembly and debugging of half-a-dozen unnecessary control cards.

From a more radical point of view, consideration might be given to the complete reorganization of computer facilities in universities. At the present time, university computer centers carefully guard their right to centralize computer facilities, in particular processors. It may be difficult to continue to uphold this policy in the 1980's. Processors are neither the most expensive components of a computer system, nor do they normally require a great deal of maintenance. Peripheral devices, such as line printers and plotters, on the other hand, account for a sizable proportion of both capital and maintenance costs of a computer system. It may be argued that centralization of these units in a computer center, accessed by processors distributed around the campus, could contribute greatly to a more flexible and user responsive computer service.

REFERENCES

1. Reiners, W. J., "Computers in Building," *Building*, Vol. 219 (6656), Dec. 11, 1970 pp. 131-44.
2. Miller, W. R., "Computers in Architecture," *Datamation*, Sept. 15, 1971 pp. 20-6.

3. Gero, J. S., "Computers and the Architecture Student," *R.A.I.A. News,* April 1970, Royal Australian Institute of Architects, Sydney, N.S.W., Australia.
4. Pohl, J. G. and J. Conrad, *User-Guide for Computers in the Architectural Profession,* 1974, Published by the authors—547 Stanford Drive, San Luis Obispo, CA 93401.
5. Chapman, A. J., *User-Guide for Architecture Students—Batch Computer Programs,* Vol. 1, 1974, El Corral, California Polytechnic State University, San Luis Obispo, CA 93407.
6. Pohl, J. G., *User-Guide for Architecture Students—Time-Sharing Computer Programs,* Vol. 1, 1974, El Corral, California Polytechnic State University, San Luis Obispo, CA 93407.



Area Director:
Donn B. Parker
Stanford Research Institute
Menlo Park, California

Making computers safer

As computer technology accelerates in use and application, the supportive functions that make the technology safe to use lags behind. Four of these functional areas are to be addressed by sessions in the conference.

The need for attention to be given to these areas has become real and threats have increased to the extent that great harm to people, organizations, and institutions has been experienced already or is imminent. This is occurring as computers proliferate into sensitive and vital parts of society. This computerization of many activities has often inherently increased the safety of performing those activities without any special effort or concern for safety. However, people and organizations are not prepared to cope with *new* hazards computer technology has created. But in all areas of penetration by the computer, at least the potential for increasing safety and well being has increased many-fold. It remains for scientists, technologists, and decision-makers to exploit that potential to elevate the safety and well-being of society to new heights.

At this conference, an opportunity to engage in information and opinion exchange on making computers safer will be provided in sessions in the areas of technological methods, good practices in computer environments, auditing and professional responsibilities.

- Technological methods of making computers safer are foreseen as the result of long term research, but the time lag between research solutions and general use is discouragingly slow.
- Good practices in computer environments is basic to the safe use of data processing facilities. Documentation of good practices such as the new *AFIPS System Review Manual on Security,* to be closely examined in this session, could be the best means of establishing general use of good practices.
- Auditing must be recognized as an essential function within the computing environment to assure management that safeguards are adequate in data processing.
- Professional responsibilities in making computers safer take many forms including certification of knowledge, licensing, establishment of competence, ethical considerations, employee responsibilities and the societal image.

All these issues make up our look at this problem of making computers safer.

Secure computer operation with virtual machine partitioning

by CLARK WEISSMAN
System Development Corporation
Santa Monica, California

BACKGROUND AND MOTIVATION

In an earlier paper,[1] I noted the extremes to which one commercial company, System Development Corporation (SDC), has gone to satisfy its disparate security/privacy requirements; this may be typical of at least one-half of the industry. In brief, the solution is to have two CPUs; six different operating systems, including two custom products; six user classes from research and development to a Corporate Management Information System; and four blocked-time periods, i.e., Periods Processing, per machine. This distasteful Periods Processing (PP) solution is the only choice for demanding facility operators, short of not processing sensitive data, and the only one accepted by the U.S. Department of Defense.

PP is economically and procedurally unsatisfactory, wasting under-utilized machine and human resources, and disruptive to normal job flow, turnaround, and personnel productivity. A sound economic solution is offered by multiprogramming systems; however, they are unacceptable because they have demonstrated vulnerability to accidental data leakage and planned intrusion.[2] Clearly, a single system that satisfies the comprehensive security requirements of (1) foiling data theft, (2) system corruption attempts, and (3) denying legitimate service by sabotage of military secrecy, government privacy, industry proprietary and individual civil rights applications is beyond today's state of the art.[3] A compromise solution between the extremes of PP and shared resource multiprogramming is needed and *is* within the reach of current technology.

The compromise solution is not relaxed standards for system security, but diminished user-process functionality. If user processes cannot share data nor inter-communicate in any manner, but can only share the physical bare hardware, an incorruptible multiprogramming system can be built. And that system will be a useful alternative to PP. It is the thesis of this paper that Virtual Machine (VM)[4] executive software is just such a compromise solution.

A VM-based system ruggedly isolates possibly hostile user processes by encapsulating them in individual VM environments that limit data sharing, interprocess (i.e., inter-VM) communication, flaw propagation, and exploitation from other VM "partitions." The strength of this thesis was tested in a recent, joint experiment by IBM and SDC, using VM/370. Methodology and results of this experiment are described elsewhere,[5] but summarized later in this paper, as the basis for a security-Hardened, No-Sharing (HNS) retrofit version of VM/370. But first, let us look more closely at the characteristics and the trade-off advantages and disadvantages of PP and VM.

PERIODS PROCESSING AND VIRTUAL MACHINE DESCRIPTIONS

In PP security, a physical security perimeter is established to encapsulate a given computer environment from external threats. All boundary crossings are carefully examined and regulated by human and machine devices. Analogously, with VMs, a security perimeter, i.e., a VM environment which restrains that process from breaking through the security perimeter of any other VM, is established around each user process. All boundary crossings are interprocess calls and context (i.e., job) switches between a VM and the executive Control Program (CP). Such crossings are regulated by software and machine devices.

Neither security scheme mentioned above is perfect: humans fail; machines fail; software fails; security fails; interlopers mount more sophisticated attacks. Each security scheme has advantages and disadvantages.

Periods processing profile

The Defense Contracts Administration's Industrial Security Manual and numerous commercial publications define good practice for PP facility operation. There are four important elements.

Physical perimeter

A physical perimeter is established to enclose the restricted area. Fences and walls are built. Special vaults are created to store sensitive media. All entry is controlled through established checkpoints. For higher degrees of security, electromagnetic radiation leakage from the area is reduced by shielding. Where denial of service threats are of concern, self-contained power, air conditioning, water, and other consumables are established and stockpiled. Finally, emergency equipment (e.g., fire retardants) are provided to detect, alarm, and treat hostile conditions.

Clearance level setup

When the restricted area is to be dedicated to a classified level, the area is procedurally raised to that level in designated steps. First, all uncleared and unauthorized personnel are removed from the area. All unclassified media are filed or inventoried and, in all cases, clearly labeled. The computer is stopped, the job stream aborted, and a ritualistic memory cleansing begun. The computer and its telecommunications configuration is changed, as necessary, to sever any direct physical link to areas outside the security perimeter. Dial-up phone equipment is disabled or switched to encryption mode. At this point, the area is sealed off and clearance is raised to "system high." The vaults are opened; classified media are removed; the classified system master is loaded from the vault-stored tape or disk library; and the classified job is run, standalone. All input media have been previously inventoried; any job-produced output subsequently is labeled and inventoried also.

Access controlled perimeter crossing

All users, I/O media, or digital communications crossing into or out of the restricted area are scrutinized for identity, authentication of identity (e.g., badge or key), and authorization (i.e., clearance). All media are labeled and logged; digital communication is encrypted.

Sanitization between time periods

After each classified job, the area must be "sanitized" for the next user by destroying all memory residue of the prior job. If the next job is also classified at the same level, clearing memory, and I/O media control is sufficient. If the area is to be downgraded, then the inverse of clearance level setup is employed. Waste is destroyed; media are logged and vaulted; memory is cleared, including the typewriter and printer ribbons, and equipment is reconfigured.

It is important to realize the inconvenience and cost of these PP procedures. Setup and sanitization times typically run 30 minutes, with the CPU idle and unusable for that time. In addition, the "uniprogramming" mandate underutilizes the CPU further, and all that CPU waste is charged to the classified user and eventually back to his employer or contract. But that is not all; further uncalculated costs are borne by that user in a 12-to-24 hour turnaround time since PP, typically, is twice a day (morning and evening). Then, there are the costs to the unclassified users who are unable to use the facility during PP, or were disrupted by PP during their prior use.

VM/370 characteristics

An alternative PP mode of computer operation, which is used at some large facilities having multiple machines available, is to dedicate the different machines to different security levels, and to operate each one at its level continuously all day. This mode saves all the set-up and sanitization time and permits some multiprogramming of like-clearance jobs. This multiple machine mode of operation is primarily the thesis of this paper in the sense that the physical machines are replaced by *virtual* machines. The virtual machines are simulated on a single-equipment configuration, thereby making the benefits of the security scheme available even to small installations.

The heart of VM/370 is the Control Program (CP) that divides the S/370 hardware, by simulation, into a multiplicity of virtual machines that are identical in program execution to the bare S/370 hardware. A VM has a virtual CPU, virtual memory, virtual I/O channels, virtual devices such as "minidisks," and virtual unit record equipment, i.e., spooling. The CP can configure the VMs differently, based on a directory-stored definition of each VM. It dynamically maps and simulates these virtual resources on the physical hardware.

Because the VM acts as a bare S/370 machine, a VM user can operate any software that runs on S/370, including applications, DMS, and other operating systems (generically noted herein as VMOS), such as: OS, DOS, VS, and even the CP itself. (This recursive VM property is of theoretical and practical R&D interest as a completeness test of a system's security architecture.) Conventional operating systems do not have this capability. The CP displays an unusually "clean" interface that is unambiguously and identically presented to each VM user process. It is conceptually simple, small, and contains little more than is necessary to simulate and allocate S/370 hardware resources equitably among the concurrently operating VMs, i.e., multiprogrammed S/370. Furthermore, unless special intercommunication provisions are made in the CP or in VM configuration definitions, each VM is an independent machine, isolated, compartmented, unaware of any other VMs, and potentially able to operate at security levels different from other VMs in a manner analogous to the physical, multiple machine, PP security mode mentioned previously.

VM and PP comparison summary

Figure 1 illustrates the salient characteristics of VM and PP modes of secure facility operation. Right and left diagonals of the Pros and Cons Matrix summarize the tradeoffs between the security schemes.

PP offers DOD acceptance, standard OS systems with the best OS performance compared to VM's restricted VMOS capabilities needed to suppress sharing features, performance loss to CP overhead, and added equipment investment, including multiple unit record gear and more disks needed to avoid sharing. However, VM can yield lower operating costs through shared-use multiprogram-

	VM	PP
PROS	• MULTI-PROGRAMMING • LOW O&M COST • SHARED COST/USER • OPS CONTINUITY	• BEST OS PERFORMANCE • LOW INVESTMENT COST • STANDARD OS CAPABILITIES • DOD SOP
CONS	• PERFORMANCE LOSS/VMOS • ADDED INVESTMENT COST • RESTRICTED VMOS CAPABILITIES • NEED ACCREDITATION	• STAND-ALONE • HIGH O&M COST • HIGH USER COST • OPS DISRUPTION

Figure 1—Salient VM and PP tradeoffs

ming and continuity of operation. The avoidance of operational disruption to set up and sanitize PP is considered the greatest asset of VM in facility manager and user convenience.

STEPS TO SECURE COMPUTER OPERATION WITH VM PARTITIONS

Current virtual machine systems are not secure. There are no Secure SubSystem (S^3) applications; systems operators, programmers, and maintenance personnel have greater knowledge and an increased opportunity to attack the system than the transaction-oriented applications users. Figure 2 assimilates these facts into a feasible long-range, four-stage strategy to secure, virtual-machine-based multiprogramming computer operation.

Stage 1: physical perimeter

Stage I establishes the physical security perimeter for PP previously described. Stage I is the important physical security foundation for the later stages, which introduce procedural and software controls to partition the physical perimeter into multi-level security environments.

Stage II: composite perimeter

The objective of Stage II is to supplement and extend the physical perimeter with procedural controls that divide the user population into two disjoint classes: system users and application users. For a given installation, no individual should be a member of both classes, nor should system and application users be permitted to use the computer concurrently. Each class should have its own PP or dedicated machine. System users should always operate with a stand-alone system to protect them from one another. However, applications users *may* be allowed to run multiprogramming if:

- Systems and program-development are prohibited.
- They are identified, authenticated, and authorized application users.
- They only use "accredited" Secure SubSystems (S^3).

The security principle embodied here is the compartmentalizing of users to reduce their interaction and to contain any leaks. Since the operating system is insecure, we force system users into separate PP compartments separate from the applications they might exploit. Application users are not familiar enough with the operating system architecture to be able to corrupt or exploit it. In addition, we compartmentalize them within an S^3 that further constrains their unauthorized actions and diminishes the risk during their concurrent system use.

The feasibility of building an S^3 is still being debated in the R&D community and beyond the scope of this paper.[3] However, technical opinion is favorable if three conditions are met:

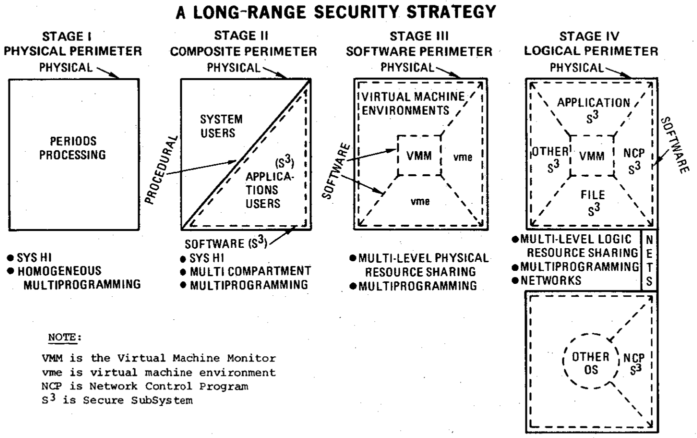
- The S^3 is a highly restrictive, transaction-oriented, formally specified command and query language application.
- The application architecture subjects each transaction to security access control checks.
- The S^3 implementation is verified as complete and correct.

Stage III: software perimeter

Stage III further extends the security perimeter with the CP, also called a Virtual Machine Monitor (VMM), software-defined virtual machine environments, as described previously. Since these VMs are non-sharing and non-communicating compartments, Stage III's system and application classes can safely be multiprogrammed in their own dedicated VMM, thereby eliminating PP altogether.

Stage IV: logical perimeter

Combining Stages I, II, and III yields Stage IV. Whereas Stage III permitted multiprogramming and sharing of the physical hardware resources, Stage IV extends the dialectic and permits the sharing of logical resources



NOTE:

VMM is the Virtual Machine Monitor
vme is virtual machine environment
NCP is Network Control Program
S^3 is Secure SubSystem

Figure 2—A long-range security strategy

for several reasons:

- The physical perimeter protects against external abuse.
- The VM protects against internal attack.
- An S^3 provides correct use of designated transaction resources.
- An S^3 within a VM within a physical perimeter can allow safe intercommunication between VMs and between accredited secure subsystems.

Once safe VM-VM communications can be assured, specialized S^3 can be constructed to share logical system resources, such as, a File S^3 or a Network Control Program (NCP) S^3.

This security strategy is logical and incrementally builds upon earlier accomplishments. It provides a road map to a secure, flexible future while maintaining secure compatibility with the past. However, it *is not* an accepted solution. DOD and other stringent security-concerned organizations await proof of its safety. Since no Stage IV systems exist, we can only assess its security potential by examining the closest facsimile Stage III system available—VM/370.

A SECURITY ANALYSIS OF VM/370

Last year, a joint project was formed between IBM Research, Yorktown Heights, New York, and the System Development Corporation, R&D Division, Santa Monica, California, to conduct experiments and empirically analyze the security of VM/370. The results are being processed for detailed publication, but some general data regarding the experimental method and results are available.[5] In this section, an overview of the issues and findings is given.

Flaw hypothesis methodology

The basic security analysis technique employed in the experiment is the SDC-developed Flaw Hypothesis Methodology (FHM), fully described elsewhere.[6] Essentially, the method asserts the "truth" of system capabilities expressed in computer code, design logic, and user manuals, and then explores counterarguments that negate the truth. A flaw is just such a counterargument and proves the truth assertion false. The flaw demonstrates that the asserted capability is really some different, unassessed capability. The resultant capability may be exploited to exceed, circumvent, or neutralize security controls. The exploitation of one or more such flaws is a system penetration.

The method emphasizes finding flaws, not exploiting them. Such penetration efforts are largely comprised of producing I/O support code to move unauthorized information in and out of the system. Comprehensive flaw finding requires three stages: generation of an inventory of suspected flaws, i.e., "flaw hypotheses," confirming the hypotheses, and generalizing the underlying system weakness for which each flaw represents a specific instance.

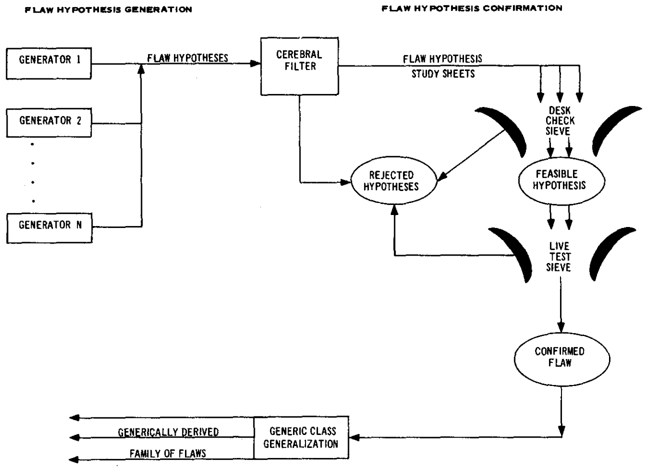


Figure 3—SDC security analysis method

Figure 3 pictorially illustrates the method via movement of a flaw hypothesis through the three stages. At the left of the figure, various "flaw generators" produce the flaw hypotheses. These generators are essentially heuristics for spotlighting system areas that have high flaw potential. The "cerebral filter" is the first of three screening sieves to confirm the flaw; it represents the collective wisdom of the analysis team. The next sieve is "desk checking," using documents and listings to prove the flaw; the bulk of the flaws are confirmed at this sieve. The last sieve is "live testing" of flaws that have a high vulnerability risk, or where complex logic is more easily probed by machine. Finally, confirmed flaws are "inductively" studied to uncover generic classes of flaws. Such classes become new flaw generators and close the analysis loop.

Experimental results: VM/370 security strengths

We found VM/370 to be securable and a potentially excellent resource sharing system, when compared to more conventional operating systems, and we believe that the virtual machine organization best suits the requirements of a multilevel security installation. This belief is based upon several findings.

The architecture of VM/370 isolates the CP and user VMs very well in three fundamental ways. First, the address space of all entities—CP and each VM—are disjoint and dynamically mapped into real memory by the Dynamic Address Translation (DAT) hardware. It was not possible to break out of a VM address space to access alien, system, or residue data by CPU programs alone. Second, the CP and VM run in different hardware operating states—S/370 Supervisor and Problem states, respectively—with privileged instructions reserved to the Supervisor state. The CP-VM interface is elegantly simple, just a privileged instruction execution trap. There are no complex system calls or parameter passing. Furthermore, the VM is not dispatched until the trap processing is completed by the CP, thereby avoiding memory restoration and asynchronous complexity. Special CP treatment is given I/O traps to allow the VM to have virtual I/O. The VM's Channel Control Word (CCW) is copied into CP memory, given a static security analysis (e.g., device address legality, memory address bounds check), and translated according to the VM's address space. This elegant solution to secure I/O prohibited most penetrations that were attempted during our experiment; however, the complexity of S/370 I/O did allow opportunities to outsmart the checking, which are discussed in the next section.

Third, VM/370 pays careful attention to formal authorization mechanisms. All users must be known to the system by ID numbers that they must authenticate by password. Satisfactory identification guarantees the user an authorized level of access, and it is only to the VM predefined for him in the VM/370 directory. This feature restricts users to preassigned environments controlled automatically by software. By extension, this feature can define the legal VMOS and application subsystem perfectly consistent with the S^3 compartmentalization called for in Stage IV of the security strategy. Finally, there is no need for on-line operator functions, operator messages, or an operator and his console, since CP only defines VMs for others to employ. (The individual VM VMOS may require an operator station, as defined by its configuration in the CP directory description, but such a station is contained in capabilities to just its VM environment and hence can be viewed as just another on-line user. This is precisely, in fact, why most VM configuration definitions equate the user and operator terminals.) The security importance of this feature (or lack of feature) cannot be overestimated when you consider the discretionary authority of the computer operator in most conventional systems. It is a standard penetration attack to inveigle the operator to give unauthorized privity. The lack of such a feature makes VM/370, operator "spoof-proof."

Experimental results: VM/370 security weaknesses

Originally, many penetration attempts were possible with VM/370. The standard release contains these flaws and is not secure. These flaws include implementation (coding) errors, design oversights, design exceptions, and compromises that depart from the sound architectural design of the CP. One such example is the "Virtual = Real" option, which effectively suppresses the CP's dynamic address translation (and hence the security isolation it affords) to allow higher performance I/O for channel programs that require dynamic address computations.

Almost all flaws require the indirect aid of a channel program, running asynchronously, i.e., overlapped, with the VM. Various I/O side effects can be directed to interfere in calculable ways with the CP-VM interface. For example, since the CP-VM interface is a response to privileged instruction traps, many of the parameters "passed" by VM to CP are "by reference" back into the VM's memory. I/O side effects can overwrite and effectively change these parameters after they are legality checked, but before they are used by the CP.

The VM/370 sharing mechanisms are another source of flaws. These include mini-disks, temporary memory, and spooling. Again, I/O programs were needed to exploit the inadequate isolation provided by these mechanisms, which must not exist or which must be designed differently in a security-hardened system.

Finally, the CP is vulnerable to unrestrained resource allocation requests that preempt real memory, devices, and I/O channels to choke the system to death. Nearly all contemporary systems are vulnerable to such denial-of-service attacks.

Though all of these flaws are ultimately software design and implementation errors and thus correctible, they reveal interesting architectural and design problems of general interest to future systems and of particular interest to VM/370 hardening efforts.

TOWARD A SECURITY-HARDENED VM SYSTEM

Until more formal mathematical proof techniques are perfected, the Flaw Hypothesis Methodology is an effective security analysis technique that should be applied to determine the security of any future security-hardened system. It is my criteria for security adequacy.

A security-hardened, no-sharing (HNS) VM/370

Currently, a hardened version of VM/370 can be obtained by repair of the generic flaws uncovered in the VM/370 security analysis experiment. Retrofit strategies are generally futile for conventional operating systems; however, our experimental evidence supports the soundness and practicality of the repair approach for VM/370.

The target objective of the HNS VM/370 would be a Stage III level of performance. All VM sharing and cooperating features can be disabled in the CP to increase compartmentalization. For example, spooling can be dropped in favor of multiple unit-record equipments, and VMs can be given dedicated disk packs to avoid mini-disk pack sharing. Decreasing the I/O vulnerabilities will be difficult, but significant progress can be made: first, by improving CCW translation to prevent self modification; second, by passing all parameters "by value," which requires the CP to copy all parameters into its protected memory space before legality checking and use. The CP already follows this good practice in most instances. Third, asynchronous attacks are prevented by imposing a very simple scheduling rule: "Dispatch a VM if and only if there is no I/O pending or already running for the VM." The performance penalty is insignificant for this non-overlap rule because overlap among VMs is still available. Fourth, threshold limits and clock timeouts can add significant preemption checks to counter resource choking attacks. Fifth, decommit the Virtual = Real feature and impose dynamic address translation for all memory references.

Finally, all design, implementation, and operational flaws must be corrected and are correctable. These changes collectively form a sound foundation for a security-hardened VM/370 that could gain DOD accreditation for multi-level secure multiprogramming.

Future VM system prospects

A more fundamental understanding of secure system architecture is needed. Issues needing clarity include: secure I/O, safety of passing parameters "by value" *versus* "by reference," the side effects of parallelism by collusive VMs, a satisfactory spooling solution, and fair, non-preemptive resource allocation schemes. Improved methods of assuring secure implementation are also required.

Work is in progress at a number of R&D organizations that are exploring these issues in concert with the virtues of virtual machines.[7] Because of its conceptual simplicity, a Virtual Machine Monitor (VMM) can be pared to its essentials and made quite small. This dual asset of simplicity and small size makes a VMM an attractive candidate for consideration by the emerging technology of formal correctness proofs. When formal correctness proofs are subsequently coupled with a good mathematical model of the security adequacy of a VMM, the VMM research comes closest to a universally acceptable solution to secure computer multiprogramming and achievement of the Stage IV secure system of the future.

ACKNOWLEDGMENTS

This report could not have been written without the enthusiastic support, ingenuity and competence of my colleagues Dick Linde and Ray Phillips at SDC, and Dick Attanasio, Les Belady, Joel Birnbaum, and Peter Markstein of IBM. I have also melded ideas from discussions with consultants Jim Anderson and Jerry Popek.

REFERENCES

1. Weissman, C., "SDC Need for a Secure Multilevel Classified Computer Facility," SDC SP-3700, March 1973. Presented at IBM Data Security Symposium, Cambridge, Mass., April 1973.
2. Branstad, D. K., "Privacy and Protection in Operating Systems," *Computer*, Vol. 6, No. 1, January 1973.
3. Anderson, J. P., "Computer Security Technology Planning Study," ESD-TR-73-51, October 1972.
4. Buzen, J. P., and U. O, Gagliardi, "The Evolution of Virtual Machine Architecture," *Proc. AFIPS NCC*, Vol. 42, June 1972, pp. 291-299.
5. Belady, L. A. and C. Weissman, "Experiments with Secure Resource Sharing for Virtual Machines," SDC SP-3769, May 1974.
6. Weissman, C., "System Security Analysis/Certification Methodology and Results," SDC SP-3728, 8 October 1973.
7. *Proc. on Protection in Operating Systems*, Institute de Recherche d'Informatique et d'Automatique (IRIA), International Colloques, Rocquencourt, France, August 1974.

The cost of computer privacy

by JEROME LOBEL
Honeywell Information Systems, Inc.
Phoenix, Arizona

INTRODUCTION

A little over one year ago, Honeywell Information Systems accepted an invitation to sponsor an important research project at Harvard University dealing with the subject of computer privacy. This work which was completed recently was performed by Dr. Robert C. Goldstein as partial fulfillment of his Doctoral Degree at Harvard.

BACKGROUND

Before looking at the results of this research project, let's examine some of the basic problems that gave vent to this important study. In particular, let's consider the meaning of the terms "computer privacy," "computer security" and "personal data."

Computer Privacy refers to the concern that an individual and/or an organization may have about who has access to their personal data. *Computer Security* addresses the need to protect all of the parts of an information system from any form of loss, destruction or unauthorized access or modification. It has been generally conceded, that the privacy of the information in a system cannot be adequately assured without the implementation of a variety of *computer security* safeguards.

On the other hand, both the Federal Government and various State Legislatures have enacted or are proposing to enact specific laws to assure that computer systems will not infringe upon the public's "right to privacy." Unfortunately, the merging of the ideal security safeguards, with the ideal legislation has not yet been achieved. The complexity of the problem is in no small way responsible for this dilemma.

As an example of this complexity, consider the HEW definition of *Personal data:* "All data that describes anything about an individual, such as identifying characteristics, measurements, test scores; that evidence things done by or to an individual, such as records of financial transactions, medical treatment, or other services; or that afford a clear basis for inferring personal characteristics, such as the mere record of his presence in a place, attendance at a meeting, or admission to some type of service institution."

Public pressure is being felt throughout almost every legislature in the land—to do something about this problem—with the preference being: "Let's not have the problem to begin with."

As an example, the October 2, 1974 issue of *Computerworld* contained an article describing the strong privacy stand taken by the *Republican Research Committee Task Force on Privacy,* which is the policy making arm of the Republican Party in the House of Representatives.

The committee recommended that "all rules pertaining to the collection of information should cover Databanks in both the public and private sector, not just governmental Databanks." On the other hand, the following statement in the committee report seems to recognize that too much regulation could also be a problem: "Despite their potential for abuse, Databanks remain an inescapable fact of life in a society growing more complex and more technological. The task force does not oppose Databanks as such, but favors strong safeguards against their misuse."

Landmark federal privacy legislation has recently been passed. The new law commonly referred to as the Privacy Act of 1974 (Public Law 93-570) includes a number of very important computer user ramifications. For instance, the new law specifies that:

1. The Government has within 10 days (excluding Saturdays, Sundays, and legal public holidays) to acknowledge in writing the receipt of a request from an individual for amendment to his records;
2. The requirement to notify publicly (in the case of the Government in the *Federal Register*) at least annually the existence and character of the system of records;
3. Agencies establish appropriate administrative, technical, and physical safeguards to insure the security and confidentiality of records and protect against any anticipated threats or hazards to their security or integrity;
4. Agencies establish procedures for the disclosure to an individual upon his request of his records or information pertaining to him;
5. Agencies establish fees to be charged, if any, to an individual for making copies of his records excluding the cost of search and review of the records.

We can already predict that major administrative difficulties and expense may come in providing records to individuals. The expense of handling the request for a copy of their record including the assurance of identity of the individual, search, copy, and mailing are only a small part of the cost. What is often overlooked is the problem of long, drawn out discussions and arguments with the individual regarding a requested change to their records. For example, about 25 percent of the people employed in credit companies are already involved in these functions under the new lending laws.

Unfortunately, we are still operating in an environment that contains more "Unknowns" than "Knowns." Here are a few examples: How many people will request a copy of their records? To what extent will the new laws trigger strictly curiosity type inquiries? How many people will now want to see records that may be associated with a refusal of credit or a job turn-down?

At the moment, it appears that only actual experience will reveal precise answers to many of these pertinent and cost related questions. Research, however, may provide us with at least some idea of the total cost ramifications of new computer privacy legislation.

It is important to note that the social advantages of computer privacy legislation have already been well researched, and are not so often the primary issues being debated. Like most social programs, however, cost will be one of the determining factors in the implementation process.

STUDY OBJECTIVES

The Goldstein Report* referred to earlier is one of the most comprehensive studies of proposed computer privacy legislation made to date. The object of the study was to evaluate the potential impact of proposed privacy legislation on computer users, Databank developers, and on the computer manufacturing industry. In addition, the study was designed to assist legislators with weighing the merits of alternative privacy bills.

This research project is particularly significant since it is the first to so thoroughly examine proposed privacy legislation in relation to associated cost ramifications. Most computer privacy research performed prior to this study focused on problem identification, and on legislation justification from mainly a moral, legal or philosophical point of view. The practical approach and research results of this study adds materially, therefore, to our knowledge of another part of the computer privacy and security problem area.

IMPACT MODEL AND APPROACH

Goldstein approached this research project into the cost of computer privacy by setting up a computer simulation model to examine the impact of a large set of possible regulations upon the personal data system of six typical computer user organizations.

RESEARCH SITES

The six research sites included:

- A system operated on behalf of a large network of hospitals.
- An on-line system operated by a State Government Agency containing identification information on people who have been arrested in that State.
- A law enforcement system operated by a State Police Group.
- A system operated by a large consumer credit organization.
- A personnel system for an organization with about 10,000 employees.
- An on-line system operated by a large casualty Insurance Company.

The model was constructed in such a way as to make it possible to test the impact of a particular set of regulations upon various sets of system characteristics. In other words, the 29 attributes* ascribed to the model could reflect a system already designed and running, or simply the specification for a proposed information system. System attributes consisted of such data elements as the number of new data subjects added per year, the average size of a record, the number of system users, etc.

REGULATORY REQUIREMENTS

The overall impact model actually consists of a number of separate models, one for each *regulatory requirement.* A *Regulatory Requirement* is a specific, operationally-oriented step that the operator of a personal data system would have to take under the provisions of one or more pending bills.

According to Goldstein, an incomplete survey earlier this year revealed that there were 45 bills on this subject pending in the US Congress, and another 50 under consideration in 23 states.

The model, however, contains 20 basic legislative proposals* for regulating personal data systems. The proposed regulations were derived from an analysis "of existing laws, a large number of pending proposals for legislation and the general computer-privacy literature."

The impact model, therefore, operates on:

- 20 Regulations
- 29 System Attributes
- 6 Typical information systems

In addition, each of the regulations would apply to any organization that maintains an administrative automated

* Honeywell Information Systems is in the process of publishing Dr. Goldstein's report "The Cost of Privacy." Copies may be ordered from: Honeywell Information Systems, 40 Guest Street, Brighton, Mass. 02135.

* Reference: *The Cost of Privacy*—Robert C. Goldstein

personal data system. The following are the regulations used in the model:

- *Data Supply Obligation Notification*
  "Any organization maintaining an Administrative Automated Personal Data System shall inform an individual asked to supply personal data for the system whether he is legally required, or may refuse to supply the data requested, and also of any specific consequences for him, which are known to the organization, for providing or not providing such data."

- *Consent for Additional Use*
  "Any organization maintaining an Administrative Automated Personal Data System shall assure that no use of individually identifiable data is made that is not within the stated purposes of the system as reasonably understood by the individual, unless the informed consent of the individual has been explicitly obtained."

- *Check Usage Authorization*
  "Access to an individual's record shall be permitted only for purposes that have been explicitly authorized by him. Authorization of particular uses would be obtained at the time the data is originally collected, and possibly, in the case of subsequently developed applications; at a later time."

- *Maintain Usage Log*
  "Any organization maintaining an Administrative Automated Personal Data System shall maintain a complete and accurate record of every access to and use made of any data in the system, including the identity of all persons and organizations to which access has been given."

- *Record Existence Notification*
  "The operator of a Personal Data System must promptly notify new data subjects of the fact that a record is being started concerning them and must notify all subjects of the existence of their records once a year."

- *Record Existence Inquiry*
  "Any organization maintaining an Administrative Automated Personal Data System shall inform an individual, upon his request, whether he is the subject of data in the system, and, if so, make such data fully available to the individual, upon his request, in a form comprehensible to him."

- *Record Uses Inquiry*
  "Any organization maintaining an Administrative Automated Personal Data System shall inform an individual, upon his request, about the uses made of data about him, including the identity of all persons and organizations involved and their relationships with the system."

- *Data Accuracy*
  "Any organization maintaining an Administrative Automated Personal Data System shall maintain data in the system with such accuracy, completeness, timeliness, and pertinence as is necessary to assure accuracy and fairness in any determination relating to an individual's qualifications, character, rights, opportunities or benefits that may be made on the basis of such data: and eliminate data from computer-accessible files when the data are no longer timely."

- *Additional Data*
  "The addition to a record of some items of information that are not currently included may be required. This would be necessary whenever the record, without the additional information, might be misleading and result in unfair decisions detrimental to the data subject."

- *Data Accuracy Inquiry*
  "Any organization maintaining an Administrative Automated Personal Data System shall maintain procedures that (i) allow an individual who is the subject of data in the system to contest their accuracy, completeness, pertinence, and the necessity for retaining them; (ii) permit data to be corrected or amended when the individual to whom they pertain so requests; and (iii) assure, when there is disagreement with the individual about whether a correction or amendment should be made, that the individual's claim is noted and included in any subsequent disclosure or dissemination of the disputed data."

- *Subject Claim Storage*
  "When there is a dispute between a data subject and the operator of a Personal Data System over whether or not a requested change is to be made in the subject's record, his claim (shall) be noted and attached to his record for subsequent dissemination. Most of the legislative proposals based on this report have additionally provided that the Databank operator may limit the subject's claim to one hundred words if assistance is provided in expressing it compactly."

- *Subject Claim Dissemination*
  "Any claim filed by a data subject shall be included whenever the disputed data is subsequently disclosed. This would require, first, the modification of all record retrieval programs to provide that the claim field be retrieved in response to any inquiry. In addition, the need to process an extra, and rather large, field will increase the amount of computer time and data transmission time associated with each transaction."

- *Retroactive Claim Dissemination*
  "Data system operators (shall) be required to send such claims or the revision of a record which obviated the need for a claim to all past recipients of a record."

- *Record Transmission*
  "Any organization maintaining a record of individually identifiable personal data, which it does not maintain as part of an Administrative Automated Data System, shall make no transfer of any such data to another organization without the prior informed consent of the individual to whom the data pertain, if as a result of the transfer, such data will become part of an Administrative Automated Data System that is not subject to these safeguard requirements."

- *Consent to Transfer Data*
  "No individually identifiable information (shall) be transferred to a system where a lesser degree of protection would be afforded without obtaining the prior informed consent of the subject."

- *Legal Process Notification*
  "Any organization maintaining an Administrative Automated Personal Data System shall assure that no data about an individual are made available from the system in response to a demand for data made by means of compulsory legal process, unless the individual to whom the data pertain has been notified of the demand."

- *Physical Security*
  "Any organization maintaining an Administrative Automated Personal Data System shall take reasonable precautions to protect the data in the system from any anticipated threats or hazards to the security of the system."

- *Employee Training*
  "Any organization maintaining an Administrative Automated Personal Data System shall take affirmative action to inform each of its employees having any responsibility or function in the design, development, operation or maintenance of the system, or the use of any data contained therein, about all the safeguard requirements and all the rules and procedures of the organization designed to assure compliance with such requirements.

- *System Assurance*
  "Any organization operating a Personal Data System must be able to ensure that its system is in compliance with all of the regulations. For this purpose, a Databank operator would, presumably, establish an audit staff responsible for monitoring the performance of the system."

- *Public Notice*
  "It is necessary to have a mechanism for publicizing the existence and characteristics of Personal Data Systems so that people will be able to determine what systems may have records on them and whether it is important to find out. Specifically, anyone proposing to establish a new system would be required to publish a notice to that effect long enough in advance to permit affected individuals a reasonable time for comment. The same type of notice would be required prior to any enlargement of an existing system. In addition, notice of the existence and character of all systems operated by an organization would be required once a year."

Not all of the proposed regulations have a representation in the impact model. The work and the outcome of the modeling are important, however, if for no other reason than the fact that this report identifies the scope of the proposed computer privacy legislation in the US and the *extremely high price* that developers of Databanks will probably have to pay to implement and operate such systems within the confines of the planned laws.

SYSTEM RESOURCES

Once each regulatory requirement was specified, Goldstein carefully defined each basic category of system resource and the related cost that might be incurred to satisfy a particular requirement. The five basic system resources or functional elements that were used in the model included:

- Manpower
- Data Storage
- Computer Processing
- Data Transmission
- Capital

Each of these is further subdivided in order to recognize differences in capabilities and cost. A distinction is also made between resource expenditures needed only at the time of system development as opposed to costs continuously incurred for operating purposes.

For the most part, the model consists of equations representing the demand of some requirement for some resource. Or as an economist might say, the equations are "production functions"; to the extent that they specify the amount of some input required to achieve some objective as a function of various system characteristics.

RESEARCH FINDINGS AND CONCLUSIONS

The conclusions derived at in the research report are extremely informative. As an example, a number of the proposed requirements would impose a nominal cost upon the computer user. Included in this category would be such regulations as:

- Check usage authorization
- Maintain usage log

- Data accuracy
- Additional data
- Subject chain storage
- Subject chain dissemination
- Data transfer consent
- Public notice

It is pointed out that even though a regulation might be very inexpensive to comply with, the difficulty surrounding its actual accomplishment might be formidable if not impractical. In addition, an analysis of the impact model results reveals that law enforcement systems would sustain the highest compliance costs of the organizations that participated in the study.

For all six test sites, initial system compliance costs varied from 11 percent to 512 percent of original total system implementation cost. Added operating costs varied from 11 percent to 222 percent of original system implementation cost. The average annual cost of compliance per subject for the six test sites varied from 57 cents to six dollars and 97 cents. The average cost per compliance transaction varied from 15 cents to 9 dollars and 64 cents.

Another conclusion that can be drawn from the previously described research study is that cost alone is no basis for an analysis of the impact of proposed privacy legislation. Cost does lend perspective, however, to the overall difficulty of compliance. In addition, it is clear that several of the most expensive regulations appear to offer the *most* protection to the individual, and therefore should not be abandoned for cost reasons alone.

For all test sites, physical security, employee training, and the data supply requirement would imply highest *conversion cost* in that order. The highest *continuing cost* regulations were:

- Record existence notification
- Data accuracy inquiry
- Record use inquiry
- Physical security
- System assurance

If one were to summarize the overall conclusions arrived at in the report, the following tend to be the most significant:

- *Government and the Public*
  First, as far as the Government and the public are concerned:
  —Is the increase in privacy worth the cost?
  ANSWER: In all probability there will be some significant reduction in business and Government efficiency. However, if the legislation is carefully worded so as to allow grace periods for compliance and exception-oriented responses (such as allowing dissemination or other use of personal data if a response to notification is not received in a specified time period) privacy legislation can be tolerable and beneficial.

- *Data Processing Industry*
  Second, what position should the data processing industry take with regard to proposed computer privacy legislation?
  ANSWER: The gist of research study conclusions in this area is that privacy legislation should constitute a challenge and opportunity to equipment suppliers. In particular, there is reference to such product opportunities as:
  —User and terminal identification devices
  —Larger and faster direct access storage devices
  —More secure computers featuring better access control mechanisms
  In addition, the computer itself may offer a solution to the control of the added manual and clerical costs that can arise from the proposed privacy legislation.

- *Data Bank Operators*
  How serious would the compliance problems be to Databank operators if the proposed privacy legislation is enacted?
  —The report certainly does not minimize the problems of technical feasibility, cost, or the additional capability that probably would be required to comply with the new regulations. Under the best conditions, compliance will be tough, and the more regulation, the tougher it may be.
  On the other hand, Dr. Goldstein has done everyone a favor by pointing out the many things that a computer user *can do* to greatly minimize the impact of the regulations.
  —Important Examples Include:
    - Paying subjects a nominal fee for the use of their data.
    - Lobby for changes to the proposed legislation to make them more livable.
    - Make certain that the data processing industry understands the need for new hardware and software products capable of handling compliance problems in a more cost-efficient way.
    - Arrange for the passing on of the additional compliance costs incurred to those people or organizations that initiate the inquiries or the addition of a personal data record.
    - Don't put so much personal data into the system to begin with.

Finally, we need to consider how important it is when discussing computer privacy problems to examine the issues through different lenses. As an example, look at how our perspective changes if we switch from the lens of a data processing or systems professional to that of John Q. Citizen. The answer to the computer privacy problem certainly relates to the amount of concern and ingenuity we apply *both* as professionals and as individuals.

In addition, have you ever stopped and asked yourself the question: What you can do in your own installation to reduce or even eliminate the need for more regulation? From a practical standpoint, the initiation of better security and internal controls would be a good place to start. After all, in more cases than not the public has a very legitimate right to complain when a system does not display reliability, integrity or fairness. In other words, the machine or the system should not be blamed for a problem that one or more of us caused in the first place.

Insuring individual's privacy from statistical data base users*

by MOHAMMED INAM UL HAQ
State University of New York at Stony Brook
Stony Brook, New York**

INTRODUCTION

"Privacy is the claim of individuals, groups or institutions to determine for themselves when, how and to what extent information about them can be communicated to others."[1] The problem of privacy arises when people are asked to provide personal information either on a mandatory or voluntary basis. The persons about whom the data are collected may be apprehensive that unscrupulous individuals could gain both economic and political advantage over them through access to their personal information. Data collecting agencies often conduct surveys to collect data to use for statistical purposes. In a mandatory survey, the respondent cannot refuse to submit the data. Still, collecting agencies realize that the real foundation of reliable statistics is public cooperation and not the threat of prosecution. In a voluntary survey, the collecting agency has no power and nothing to offer but the general benefits that can accrue to the public from the information.[2] It promises respondents that data will only be disclosed for statistical purposes. In both cases, the collecting agencies can better accomplish their aim if they can assure the respondent that their privacy will be respected. Many agencies use computers to store collected data. The computer provides a central source of information available to many different users. In addition, the information can be retrieved very quickly. However, there are many problems which are associated with the security of computerized information such as: validity of the user, clearance of system personnel and safety for masquerading, electromagnetic pick up, wiretapping, browsing, piggy back entry, etc.[3,4,5,6] Even if these problems did not exist, and assuming that the collecting agencies have good intentions and intend to allow their users to ask only statistical queries about the stored data, their promises are not sufficient to guarantee that there will be no disclosure of any individual's personal information. Hoffman and Miller[7] have shown that a user can combine the answers to some specific statistical queries and his previous knowledge about an individual's personal information to find out more about that individual. Therefore the problem considered is to determine the conditions which guarantee that a user who is allowed to ask only statistical queries cannot be successful in obtaining any more information about any individual than he already has. Before suggesting a solution to this problem, some illustrations to show how disclosure may occur and some definitions are reproduced from an earlier paper by the author.[8]

DEFINITIONS

A characteristic is a piece of information. It has a name and a value. For instance, (Name, Zalkind) and (Age, 26) are characteristics. Let N_1, N_2, . . . , be names and for each N_i there exist values V_{i1}, V_{i2} . . . . A characteristic C_{ij} is a pair (N_i, V_{ij}). A family of characteristics F_i is a set of all characteristics having the same name N_i.

The group of persons about whom the information is collected is called data persons. A record r_i is a set of all characteristics collected about a data person p_i. A data base is a set of records of all data persons.

A query is a statistical question which can be asked about the data base. A query is formed by taking the conjunction, disjunction and negation of characteristics. These operations are denoted by $\wedge$, $\vee$, $-$. Formally a query is defined as:

(1) A characteristic standing alone is a query.
(2) If q is a query then—(q) is a query.
(3) If q_1 and q_2 are queries then $(q_1 \vee q_2)$ and $(q_1 \wedge q_2)$ are queries.
(4) Nothing else is a query.

Where no confusion arises, parentheses will be eliminated.

The retrieved set R_q of a query q is the set of all data persons who have those characteristics in their records which are implied in q. The semantic of a query q is, because of a statistical data base, "How many data persons have all the characteristics which are implied in q?" The answer to the query q, denoted by A_q, is the cardinality of the retrieved set R_q. A data base together with a procedure for limiting the access to the data base for statistical purpose will be referred to as the "System". A query is a permitted query if it is permitted by the system; otherwise it is a restricted query.

* This work was supported by National Science Foundation Grant # GJ 28177 while the author was a student in Department of Computer Science.

** Present address: High Impact Anti-Crime Program, 38 Halsey Street, Newark, New Jersey

A user of the system may obtain information about some of the characteristics of some data persons by using mechanisms not involving the system. For example, a user may know the town in which a particular data person lives. Such information plays an important role in the security of the data base if the data base contains characteristics with the same name "Town".

A user's supplementary knowledge is a set of information about the data base and all other information which can be derived from it that a user knows from a source other than the system. It consists of the following:

(1) The set of supplementary queries (Q_s): It is the set of queries whose answers are known to a user. An element of this set is a supplementary query.
(2) Supplementary data base: It is the set of all known records, where a known record is a set of all characteristics of a data person which are known to a user.
(3) The set of supplementary properties: It is the set of properties of the data base which are known to a user.

An example of a supplementary property is that a user may know that the answer to a query q_1 is the sum of the answers of the queries q_2, q_3 and q_4. In essence, a user may know relations which will be helpful in finding the answer to a restricted query.

Disclosure is defined as: Disclosure occurs if a user can enlarge his supplementary knowledge by adding some information about the data base which is not permitted by the system.

Statistical disclosure occurs if a user can enlarge his set of supplementary queries, by adding a restricted query.

Resultant disclosure occurs if the answer to a restricted query can be computed from the answers of permitted queries by using the properties of the data base which are known to a user. Resultant disclosure is a special case of statistical disclosure in which the set of supplementary queries is empty.

Positive personal disclosure occurs if a user can enlarge his supplementary data base by adding a characteristic to some record.

A user of the system may know from his supplementary knowledge that a data person does not have a particular characteristic. If a data person does not have a characteristic C_{ij}, then it will be referred to as that he has a negative characteristic which will be denoted by $-C_{ij}$. Clearly a negative characteristic is a piece of information. In fact, knowledge of negative characteristics coupled with the property of the family of characteristics may be used to obtain a positive personal disclosure. Hence, a record in a supplementary data base may also contain negative characteristics (negative characteristics are not included if they are redundant).

Negative personal disclosure occurs if a user can enlarge his supplementary data base by adding a negative characteristic which is not redundant, to some record.

Personal disclosure is said to occur if positive personal disclosure and/or negative personal disclosure occurs.

EXAMPLES OF PERSONAL DISCLOSURE

Example 1

Let q = (Soc. Sec. #, SS_{116}), $A_q=1$. Let the user's supplementary knowledge include that he knows a person p whose Soc. Sec. # is SS_{116} and the fact that Soc. Sec. # is unique for each data person. As $A_q=1$, there is a personal disclosure that p is a data person.

Example 2

Let q_2 = (Major, Math.) $\wedge$ (Marital Status, Married), $A_{q_2}=0$. Therefore there is no data person who is majoring in Math. and is married. Knowing that the only alternative of married is single and if a user knows from his supplementary knowledge a data person who is majoring in Math.; he can conclude that the person is single.

Example 3

Let q_3 = (Major, Comp. Sc.) $\wedge$ (Sex, Female), $A_{q_3}=5$. Let q_8 = (Major, Comp. Sc.) $\wedge$ (Sex, Female) $\wedge$ (Housing Status, On Campus), $A_{q_8}=5$. Therefore a girl who is majoring in computer science lives on campus. Hence, if a user knows a girl who is majoring in computer science, then he can conclude that she lives on campus. Therefore there is a personal disclosure.

EXISTING SOLUTIONS

Reed[9] has suggested a method to keep individual records protected by distorting the information in records which are available for statistical summaries. His method does not insure security for personal disclosure in cases such as illustrated in the examples. Hansen[10] has suggested a method which decreases the chances for disclosure by giving an approximate answer to a query rather than exact number. He also mentioned an approach in which the answer to a query is computed on a sample data and then multiplied by the sampling factor. These methods decrease but do not eliminate the chances for disclosure. Fellegi[11] has suggested an algorithm to check the occurrence of a personal disclosure. Not only is his method very cumbersome, but it does not work in all cases as it stands.[8] The necessary and sufficient conditions for statistical and positive personal disclosure in a specific case, have been given in an earlier paper.[8] We are now suggesting a solution to this problem by defining the conditions which are necessary and sufficient for statistical disclosure in a general case, and also for positive and/or negative personal disclosure in a specific case.

STATISTICAL DISCLOSURE

It is obvious that some of the queries cannot be permitted if personal disclosure is to be prevented. But if some of the queries are not permitted then it becomes necessary to assure that the answer to a restricted query cannot be computed from the answers of permitted queries. The following theorem provides the necessary and sufficient conditions for the answer of a query to be computed from the answers of other queries.

A query q, in general, partitions the set of all data persons into two sets, namely the set of those data persons who are included in the retrieved set of q, and the set of those who are not. Therefore, a set Q of n queries $q_1, q_2, \ldots, q_n$ partitions the set of all data persons into 2^n disjoint sets where a typical set contains all those data persons who are simultaneously included in the retrieved sets of k out of the n queries and not included in the retrieved sets of the other $n-k$ queries in Q. One of these sets is the set of all data persons who are not included in the retrieved set for any $q \in Q$. Excluding this particular set then there are 2^n-1 sets called the partition sets induced by Q. The set of partition sets induced by Q is denoted by P_Q and will be referred to as the set of partition sets. A partition set is denoted by p. Let $L=|P_Q|$ where $|P_Q|$ is the cardinality of the set P_Q. If $q \in Q$ then the retrieved set of q is the union of some (Possibly all) of the partition sets; therefore $A_q=c_1|p_1|+c_2|p_2|+\cdots+c_L|P_L|$ where for all i, c_i is either zero or one. Also, if p is a partition set, then p is a retrieved set of a query which is a conjunction of queries q_i^* such that either q_i^* or the negation of q_i^* belongs to Q.

A set of queries Q is a dependent set if there exists a query $q_i \in Q$ such that the answer to q_i can be computed from the answers of the queries which belong to Q; otherwise it is an independent set. A dependent set is a minimal dependent set if no proper subset of it is a dependent set. We may refer to minimal dependent set as a dependent set.

Theorem I: The answer to query q' can be computed by a user from the answers of the independent set Q of queries if and only if:

(i) $P_Q = P_{QU\{q'\}}$ and is known to a user.
(ii) $|Q|=|P_Q|$

Proof: See Appendix I.

The theorem means that the answer to a query q' can be computed from the answers of the independent set Q of queries if and only if the query q' does not induce a new partition which is not induced by the set Q of queries, and the number of such queries is equal to the number of the partitions induced by Q. Hence, if a query q is not permitted and induces a new partition which is not induced by the set of permitted queries then the answer to the query q cannot be computed from the answers of the permitted queries.

A user may know that the answer to a query cannot be greater than some number and/or he may know that it cannot be less than some other number. The lower bound of a query q denoted by L_q is defined as the greatest non-negative integer, n, known to a user such that $A_q \geq n$. Similarly the upper bound of a query q denoted by U_q is the smallest non-negative integer, n, known to a user such that $A_q \leq n$. Let a_q denote the number of data persons who a user knows are included in R_q.

Theorem II: There is a statistical disclosure if and only if there exists a restricted query q' and an independent set Q of queries such that the following hold:

(i) $P_Q=P_{QU\{q'\}}$ and is known to a user.
(ii) There is a unique integer solution to the set of equations $\sum_{\forall p_j \in P_Q} b_{ij}|p_j|=B_{q_i}$, $\forall q_i \in Q$ where $L_{q_i} \leq B_{q_i} \leq U_{q_i}$ and $b_{ij}=1$ if $p_j \subseteq R_{q_i}$ and zero otherwise.

Proof: See Appendix II.

Example 4

Consider a set Q of queries q_i, $i=1$ to 5. Let the retrieved sets of q_2, q_3, q_4 and q_5 be the disjoint partitions of the retrieved set of q_1. Let a user know that $a_{q_1}=14$, $U_{q_1}=16$, $a_{q_2}=2$, $U_{q_2}=4$, $A_{q_3}=3$, $A_{q_4}=4$, $A_{q_5}=5$. It can be easily seen that condition (i) is satisfied but condition (ii) does not hold since there are three sets of values for q_1 and q_2, namely (16, 4), (15, 3), (14, 2) which satisfy the equations. Hence, there would not be statistical disclosure. However, if for instance either $a_{q_1}=16$ or $a_{q_2}=4$ then these equations have a unique solution. And hence, there would be statistical disclosure. In these cases, there are unique values for q_1 and q_2 which satisfy condition (ii).

PERSONAL DISCLOSURE

The examples of disclosure show that some of the queries if permitted would cause personal disclosure. The following theorems explore the conditions under which a query or a set of queries can cause personal disclosure. Some definitions are needed:

A query is a simple query if it is formed either by a single characteristic or by taking the conjunction of characteristics. A query q_i is a refinement of a query q_j if there exists a query q_k which contains at least one characteristic that appears in q_i but not in q_j, such that $q_i=q_j \wedge q_k$. Furthermore if q_k contains only one characteristic, then it is said to be an elementary refinement. Consider a query $\hat{q}$. Let no characteristic which appears in $\hat{q}$ belong to the family of characteristics $F_i=\{C_{i1}, C_{i2}, \ldots, C_{in}\}$. Let $\underline{Q}$ be the set of queries derived from the query $\hat{q}$ by refining it such that $q_j=\hat{q} \wedge C_{ij}$ for $j=1$ to n. The set $\underline{Q}$ is said to be a set of elementary refined queries of $\hat{q}$ and the set $\{\hat{q}, q_1, q_2, \ldots, q_n\}$ is said to be an elementary family of queries. A family of queries can be recursively defined as follows:

The set Q of queries is a family of queries if either it is an elementary family, or it is obtained from a family Q' by replacing a query $q \in \underline{Q}'$ with the set of its elementary refined queries.

The common ancestor query $\hat{q}$ of the family Q is denoted by $\hat{q}$ and the set of all its descendent queries from the family Q is denoted by $\underline{Q}$. A family of queries $Q = \{\hat{q}, q_1, q_2, \ldots, q_n\}$ is said to be a disjoint family if the sets $R_{q_1}, R_{q_2}, \ldots, R_{q_n}$

form disjoint partitions of the set $R_{\hat{q}}$. The following theorems are based on the following assumptions:

(1) A user is allowed to ask only simple queries.
(2) The only relation among queries which holds for a data base is the family relation and this relation is disjoint.
(3) User's supplementary knowledge is:

  (i) A supplementary data base.
  (ii) A set of supplementary queries.
  (iii) Knowledge of the fact that each data person has one and only one characteristic from a family of characteristics.

The following lemma asserts that the conditions which guarantee that no negative personal disclosure is possible, are stronger than the corresponding conditions for positive personal disclosure.

Lemma: For a data base in which a data person can have one and only one characteristic from a family of characteristics, there is no positive personal disclosure if there is no negative personal disclosure.

Proof: Clearly if a user determines that a data person $p \in R_{qi}$, then he can conclude that $p \notin R_{qj}$, where q_i and q_j belong to the same set of elementary descendent queries. The existence of such a query q_j is guaranteed, else there cannot be a positive personal disclosure. This proves the lemma.

The following definitions and notations are needed. A data person p is said to be a candidate if a user knows that $p \in R_{\hat{q}}$ and does not know whether or not he is included in R_{qi}, where q_i is a direct descendent query of $\hat{q}$. Let a_{q-qi} denote the number of data persons who a user knows are included in R_q and not included in R_{qi}. Let Q_k denote the set of all the queries whose answers are known to a user. It can be seen that there is a negative personal disclosure (i.e., a user can add a negative characteristic in supplementary data base by using the results obtained from the system) if and only if there exists an elementary family $Q = \{\hat{q}, q_1, q_2, \ldots, q_n\}$ and a data person p such that the user knows from his supplementary knowledge that $p \in R_{\hat{q}}$ and can conclude that $p \notin R_{qi}$, where $q_i \in \underline{Q}$.

Theorem III: There is a negative personal disclosure if and only if there exists an elementary family $Q = \{\hat{q}, q_1, q_2, \ldots, q_n\}$, such that the following hold:

(i) $a_{\hat{q}} > a_{\hat{q}-qi}$
(ii) $A_{qi} = a_{qi}$
(iii) $q_i \in Q_k$

Proof: See Appendix III.

The theorem states that there is a negative personal disclosure if and only if there exists an elementary family Q and a query $q_i \in \underline{Q}$ such that there is a candidate in the family and the user knows all the data persons who are included in the retrieved set of q_i. From the theorem, it becomes obvious that a user's supplementary knowledge plays a very important role in generating personal disclosure. Especially if a user has no supplementary knowledge about any data person (i.e., his supplementary data base is empty), then all those queries which do not disclose the identity of a data person can be permitted, irrespective of their answers and still there will be no personal disclosure.

Example 5

Consider example 2. Let $q_{2'} =$ (Major, Math.) $\wedge$ (Marital Status, Single), and $q_7 =$ (Major, Math.) Let $Q = \{q_7, q_2, q_{2'}\}$ be the disjoint family in which q_7 is the ancestor query. Since in example 2, $a_{q7} = 1$, $a_{q2} = 0$, condition (i) of the theorem is satisfied. Also since $A_{q2} = 0$, conditions (ii) and (iii) are satisfied. Hence, there is a negative disclosure that the data person is not married.

Let Q_p denote the set of all permitted queries. Let M be the maximum number of data persons who a user knows are included in the data base. The following theorem provides sufficient conditions for no personal disclosures:

Theorem IV: There is no personal disclosure if the following hold:

(i) $q \in Q_p \Rightarrow A_q \geq M$.
(ii) There is no statistical disclosure.

Proof: See Appendix IV.

Theorem IV may be used for implementing a simple model for a secure system. M is introduced as a maximum number of data persons known to a user. In a practical situation, it is very hard to choose M. However, it may be desirable to assume an upper limit for it. Once M is chosen and assuming that the user's set of supplementary queries is empty, a set of queries whose answers are not less than M may be selected for a possible set of permitted queries. From this set, its largest subset is chosen which guarantees that there will be no resultant disclosure if this set is permitted. Theorem IV then insures that in permitting this set of queries there will be no personal disclosure.

CONCLUSION

The theorems which are proved in this paper provide a means to check whether or not a statistical information retrieval system is secure from personal disclosure. It should be noted that in a data base in which the disjoint family relation holds, if a query is not permitted then at least one of its direct descendent queries cannot be permitted either. Otherwise, a user could add the answers to all of the direct descendent queries and find out the answer to the original query. In turn, one of the descendent queries of that descendent query must be restricted, and so on. Thus, if a query is not permitted then that query will propagate at least one query in each of its descendent families which cannot be permitted. Hence, the number of restricted queries grows very rapidly.

ACKNOWLEDGMENTS

I am thankful to Professor A. J. Bernstein and Professor J. Heller for advising me in this work. Professor Bernstein also helped me in verifying proofs of the theorems. I am indebted to Professor R. B. Kieburtz as well as Professor H. Gelernter and Professor D. H. Tycko for their continuous encouragements and sympathetic behavior with me. My special thanks are for Mr. Alan Zalkind, Executive Director, High Impact Anti-Crime Program, who gave me the opportunity to present this paper.

REFERENCES AND BIBLIOGRAPHY

1. Westin, A., *Privacy and Freedom*, Atheneum, New York, 1968
2. *Federal Statistics*, Chapter 6, Vol. I and Chapter 5, Vol. II.
3. Hoffman, L. J., "Computer and Privacy: A Survey," *Computing Surveys*, Vol. I, Number 2, June 1969.
4. Petersen, H. E., and R. Turn, "Systems Implications of Information Privacy," *AFIPS Conference Proceedings*, 1967 Spring Joint Computer Conference, pp. 291-300.
5. Turn, R., and H. E., Petersen, *Security of Computerized Information Systems*, Rand Corporation Technical Journal, The Rand Corporation, Santa Monica, California.
6. Ware, W., "Security and Privacy in Computer Systems," *AFIPS Conference Proceedings*, 1967 Spring Joint Computer Conference, pp. 279-282.
7. Hoffman, L. J., and W. F., Miller, "Getting a Personal Dossier from a Statistical Data Bank," *Datamation*, May 1970, pp. 74-75.
8. Haq, M. I., "Security in a Statistical Data Base," *Proceedings of the American Society of Information Science*, 1974 Conference, Vol. 11, pp. 33-39.
9. Reed, I. S., "Information Theory and Privacy in Data Banks," *AFIPS Conference Proceedings*, 1973 National Computer Conference, pp. 581-587.
10. Hansen, M. H., "Insuring Confidentiality of Individual Records in Data Storage and Retrieval for Statistical Purposes," *AFIPS Conference Proceedings*, 1971 Fall Joint Computer Conference, pp. 579-585.
11. Fellegi, I. P., "On the Question of Statistical Confidentiality," *Journal of the American Statistical Association*, March 1972, Vol. 67, Number 337, pp. 7-17.
12. Boruch, R. F., "Security of Information Processing—Implications from Social Research," *AFIPS Conference Proceedings*, 1972 Fall Joint Computer Conference, pp. 425-433.
13. Rothman, S., "The Protection of Privacy and Security in Criminal Offender Record Information Systems," *AFIPS Conference Proceedings*, 1972 Fall Joint Computer Conference, pp. 423-424.
14. Turn, R., and N. Z., Shapiro, "Privacy and Security in Data Bank Systems—Measures of Effectiveness, Costs, and Protector-Intruder Interactions," *AFIPS Conference Proceedings*, 1972 Fall Joint Computer Conference, pp. 435-444.
15. Comber, E. V., "Management of Confidential Information," *AFIPS Conference Proceedings*, 1969 Fall Joint Computer Conference, pp. 135-143.
16. Peters, B., "Security Considerations in a Multi-Programmed Computer System," *AFIPS Conference Proceedings*, 1967 Spring Joint Computer Conference, pp. 283-286.

APPENDIX I—PROOF OF THEOREM I

Proof of if part: Let $Q=\{q_1, q_2, \ldots, q_n\}$ and $P_Q=(p_1, p_2, \ldots, p_L\}$. Condition (i) implies that $A_{q'}=\sum_{j=1}^{L} c_j \cdot |p_j|$ where c_j is either zero or one and is known. $A_{q'}$ can, therefore, be computed if for all $c_j \neq 0$ the $|p_j|$'s are known. Also

$$b_{i1}|p_1|+b_{i2}|p_2|+\cdots+b_{iL}|p_L|=A_{qi} \qquad (1)$$

where for all $1 \leq i \leq n$ and $1 \leq j \leq L$, b_{ij} is known. Condition (ii) implies that $L=n$. As these equations are independent, there is a unique solution for $|p_j|$'s. Hence, $|p_j|$'s are known. Therefore the answer of q' can be computed.

Proof of only if part: Suppose that the answer of q' can be computed from the set Q of independent queries and condition (i) and/or condition (ii) does not hold. Condition (i) does not hold means that $P_Q \neq P_{QU\{q'\}}$. Then $P_{QU\{q'\}}$ is formed by partitioning at least one of the partition sets, say p_j, into $p_{j'}$ and p_j'', where $p_{j'}$ is contained in the retrieved set of q' and p_j'' is not contained in the retrieved set of q'. Consider a modification to the data base in which a single person who is included in $p_{j'}$ of the original data base is moved to p_j'' in the modified data base. Since the information known to the user i.e., A_{q_i}'s remains the same, he would conclude the same answer for q' which would be incorrect. Therefore condition (i) is necessary. Condition (ii) does not hold implies that $L \neq n$, where $n=|Q|$ and $L=|P_Q|$. If $n>L$, then L of the simultaneous equations (1) can be solved to find $|p_j|$'s, for $j=1$ to L. The remaining A_{q_i}, $L+1 \leq i \leq n$ can, therefore be computed from the answers of $q_1, q_2, \ldots, q_L$. Thus, Q is not a set of independent queries which is a contradiction. If $n<L$ then the set of simultaneous equations (1) do not have a unique solution. Hence, the answer of q' cannot be computed uniquely. This completes the proof of the theorem.

APPENDIX II—PROOF OF THEOREM II

Proof of if part: It can be shown as in the proof of theorem I that if condition (ii) holds then $|p_j|$'s are uniquely determined. From theorem I, condition (i) implies that the answer of q' can be computed; hence, there is a statistical disclosure.

Proof of only if part: Let there be statistical disclosure about the set Q' of queries. Let $q' \in Q'$. This part is proved by using the method of proof by contradiction. In order to obtain the contradiction, a modified data base is designed from the original data base such that the modified data base is also a valid data base and both data bases are indistinguishable to a user. Now assume that condition (i) and/or condition (ii) does not hold. If condition (i) does not hold then the same arguments as those used in the proof of theorem I can be given which lead to a contradiction. Hence, condition (i) is necessary. If condition (ii) does not hold, then at least two sets of values for $|p_j|$, $j=1$ to L, exist which satisfy condition (i). Consider a modification to the data base by altering the answers of set Q' of queries by changing the values of $|p_j|$'s and keeping all the information the same. Consequently, the user will conclude the same answers for the set Q' of queries which would not be correct. Hence, there is a contradiction. This completes the proof of the theorem.

APPENDIX III—PROOF OF THEOREM III

Proof of if part: Condition (i) implies that there is at least one candidate in family Q. Let that person be denoted by k. Conditions (ii) and (iii) imply that the user knows all the data persons who are included in R_{q_i}. Therefore he can conclude that $k \notin R_{q_i}$. Hence, negative personal disclosure occurs.

Proof of only if part: Let there be a negative personal disclosure about a person p, i.e., there exists q_i such that the user can conclude that $p \notin R_{q_i}$, given that he already knew that $p \in R_{\hat{q}}$, where $\hat{q}$ is the ancestor query of q_i. Let $Q = \{\hat{q}, q_1, q_2, \ldots, q_n\}$ be an elementary family which contains $\hat{q}$ and q_i. Hence, there is a candidate p such that $a_{\hat{q}} > a_{\hat{q}-q_i}$. Therefore condition (i) is satisfied. Now assume that condition (ii) does not hold, i.e., $A_{q_i} > a_{q_i}$. This implies that there exists a data person, say U_1, who is not known to the user, and $U_1 \in R_{q_i}$. The fact that negative personal disclosure has occurred implies that there is at least one data person, say U_2, who is included in one of the sets $R_{q_1}, R_{q_2}, \ldots, R_{q_{i-1}}, R_{q_{i+1}}, \ldots, R_{q_n}$, and the user concluded that $p \equiv U_2$ (i.e., that data person p is the same as data person U_2). Assume that $U_2 \in R_{q_j}$, $j \neq i$. Consider a modification to the data base in which two data persons, U_1 and U_2, who are included in the retrieved set R_{q_i} and R_{q_j} in the original data base, are now moved to R_{q_j} and R_{q_i} respectively in the modified data base. As the information known to the user remains the same, he would still conclude that $p \notin R_{q_i}$ which would be incorrect. Hence, there is a contradiction. Therefore condition (ii) is true. Now assume that condition (iii) does not hold, i.e., $q_i \notin Q_k$. There exists at least one more query, say q_j, from the set Q such that $q_{j'} \notin Q_k$; otherwise the user can find the answer of q_i from the linear relation which exists in a family of queries. Consider the two cases. $q_{j'}$ is $\hat{q}$ or $q_{j'} \in \underline{Q} - \{q_i\}$. In each case, a modification to the data base can be made by moving the data person U_2 to R_{q_i} and varying the answers of queries from the family Q such that the information known to the user does not change. Therefore, the user would conclude the same result, that data person $p \notin R_{q_i}$ for the modified data base, which would be incorrect. Hence, there is a contradiction. This completes the proof.

APPENDIX IV—PROOF OF THEOREM IV

Proof: As the conditions for no negative personal disclosure are stronger than positive personal disclosure, the theorem is proved for negative personal disclosure. Let Q be an elementary family which contains q_i. Assume that there is a negative personal disclosure that a data person $k \notin R_{q_i}$, given that the user already knew that $k \in R_{\hat{q}}$. By theorem III, then $a_{\hat{q}} > a_{\hat{q}-q_i}$ and $A_{q_i} = a_{q_i}$ and is known to the user. There can be two cases:

1. $A_{q_i} < M$.
2. $A_{q_i} \geq M$.

Consider case 1. As there is no statistical disclosure, $Q_p U Q_s = Q_k$, where Q_s is the set of supplementary queries. Hence, if condition (i) is true then for all $q \in Q_k$ if $A_q < M$ then $q \in Q_s$. Consequently, there is no negative personal disclosure since all the information is known to the user from his supplementary knowledge. Consider case 2. By the definition of M, a_{q_i} can at most be equal to M. By the condition (ii) of theorem III, A_{q_i} is therefore equal to M. But then condition (i) of theorem III does not hold. Hence, there can be no negative personal disclosure. Therefore, the theorem is proved.

Computers, security, and the audit function

by NORMAN R. NIELSEN
Stanford Research Institute
Menlo Park, California

INTRODUCTION

As the role of computation spread in the world of commercial data processing and in the world of organizational administration, concern and attention were directed toward such matters as programming techniques, computer capabilities, and computer performance measurement and evaluation. However, because computers became increasingly essential to the conduct of an organization's daily activities, and because these systems frequently controlled vast amounts of organizational resources, the computer became an object of attack. In some cases the organization was the target; the computer was the means to attack it. In other cases personal gain was the goal; the computer was merely the means for misappropriating corporate assets. Parker[1] has chronicled more than two hundred cases of computer-related crimes. Therefore, concern has recently been focused on the security of computer installations and the maintenance of computer system integrity. In 1974 both AFIPS[2] and the National Bureau of Standards[3] have published guidelines for computer security.

The organizational auditor has long been involved in one fashion or another with the auditing of applications processed in whole or in part by computer systems. Until recently, though, his concern has been with the application itself, not with the computer system. Great faith was maintained in the basic integrity of computer systems. Now, however, the growing incidence of computer-related crimes has shattered that faith. The auditor must also concern himself with the basic security of the organization's computer facilities and with the means by which the integrity of the systems is maintained. Thus, the auditor is rapidly being thrust into an area in which he has traditionally had little expertise or responsibility.

COMPUTER SECURITY

Four developments have served to bring concern about computer security to the forefront today. It is these developments that are rapidly pushing the auditor into the midst of the computer security problem.

The growing role of computing

Computers are becoming an integrated part of the operations of many organizations. Computer systems schedule daily work flows and product assembly sequences in manufacturing. In the transport industry, reservations, traffic movement control, and crew and equipment scheduling are computer-based. Banks are critically dependent upon overnight computer processing, so that they may open their doors the following morning. Governmental organizations rely upon computers for the disbursement of funds and the maintenance of records concerning their constituent citizens.

The computer has thus become a vital cog in most organizations. The period of time during which an organization could continue to exist in the face of the cessation of computer processing will, of course, vary from organization to organization. However, the time periods are all short in relative terms, and these periods are tending to become shorter every year. Thus, security as related to the continued availability of these computer systems to serve their organizations takes on particular importance.

Computer files as assets

Computerized files of data are in many organizations becoming synonymous with the assets recorded in them. Thus, a bank's files of account balances are just as much "cash" as the depositors' actual funds. A few bits changed in a file can yield as much real money as an armed robbery. The same holds true for payment systems associated with accounts payable, payroll, claims, dividends, pensions, and credit account balances in other organizations.

Even computerized files that do not represent cash assets are frequently valuable organizational assets in need of protection. Mailing lists are a classic example of this type of asset and have already been the target of thefts.

Privacy

Privacy is a distinct concept that must not be confused with security. However, the privacy of data cannot be assured on an insecure system. That is, a secure computer system is a necessary, but definitely not a sufficient, condition for maintaining the privacy of personal information stored in computer files.

Today there is a great deal of interest in the privacy of the individual and of data relating to the individual.[4] Bills concerned with privacy have been introduced in both the House and the Senate during the past legislative session, and numerous bills have been introduced in the various state legislatures. Although very little of this legislation has yet become law, it is clear that additional legislation will be forthcoming in this area. While the immediate impact of new laws is likely to focus on privacy considerations per se, there will necessarily be a derivative impact on computer facility security and system integrity requirements.

EXPANDED AUDIT ROLE

The fact that the security of a computer facility can no longer be taken for granted and the integrity of a computer system can no longer be assumed is leading to a much broader role for auditors in an organization's EDP operations. Considered below are six areas where much greater auditor participation is now required.

Physical security

Because of the critical role played by computing in most organizations, the auditor must take steps to verify that appropriate levels of physical security surround the computer facility. Consideration must be given to fire detection and extinguishing equipment, to pumps and alarms, to air conditioning equipment and air intakes, to power cables and communication lines, to window areas and wall construction, to activities and hazards in neighboring buildings and adjacent floors, and to cleanliness of operating areas.

Procedures (and training in the use of those procedures) are just as important as physical facilities. Personnel must know how to respond in an emergency, to protect the safety of employees as well as to protect equipment and data files. Emergency equipment must be tested on a regular basis. Clearly, it is not the auditor's responsibility to develop procedures and to carry out such tests. However, he must determine the adequacy of the equipment and the procedures that have been developed for safeguarding the facility, and he must verify that the procedures are being adhered to properly.

Access control

An organization's computer facility cannot be sealed off from the outside world, for a variety of personnel and materials must regularly move in and out of it. Again, the auditor must be concerned with the adequacy of the procedures for controlling such movements and with the verification that these procedures are being followed faithfully.

The most obvious area of concern is the movement of personnel. This includes not only regular employees, but also maintenance personnel, temporary employees, visitors, deliverymen, and so on. Further, unrestricted movement to all areas within a controlled area is unlikely to be necessary or desirable, requiring the establishment of proper access controls within the basic controlled area.

Controlling the movement of materials in and out of the controlled area may at first glance seem straightforward. However, there are many subtleties. Bombs have been shipped into computer rooms in boxes thought to contain paper for the printers. Waste and spoiled outputs hauled away in the trash have often yielded valuable or sensitive information to scavengers. Tapes or printed reports can easily be concealed in a briefcase or on the person of an individual leaving a facility.

Changes in computing technology in the past few years have added a new dimension to access control. It used to be that data inputs were physically shipped to the computer facility from the user departments or branches. Authenticating the input materials and controlling the distribution of outputs was relatively easily accomplished. Now, however, much of the input and output process is conducted over telecommunication lines. Ensuring the identity of incoming requests is a much more difficult task. This is true even when dedicated rather than dial-up telephone lines are being used. It is not surprising that a variety of research, such as that of Evans, Kantrowitz, and Weiss;[5] Purdy;[6] and Friedman and Hoffman[7] has been conducted in this area.

Operating procedures

A variety of operating procedures have a direct impact on the security and integrity of an organization's computer facility and data files, so the auditor must extend his concern into this area as well.

Separation of duties has long been a key tenet in the manual handling of financial transactions. However, the extension of this concept to cover the automatic handling of transactions within data processing systems has not always been made.

Further, the data processing operation itself requires some additional types of separation that have heretofore been foreign to the auditor. Production run scheduling and control, computer operations, and data library responsibilities must be separated. Even in environments where a single employee is quite capable of performing these duties himself, the combination of responsibilities must be avoided. Many of the cases of computer abuse studied by Parker[1] resulted from an uncontrolled combination of responsibilities.

The procedures that must be examined by the auditor can become fairly technical in the operations area. For example, the safeguards over the improper or erroneous scratching of data files can be critical, as can the procedures controlling the rotation of tapes to off-site backup storage facilities. Operating concerns even extend into the computer system itself, covering such items as file access protection from unauthorized requests, tape label recognition to guard against operator volume-mounting errors, and memory protection features in multiprogrammed systems.

Programming

Programming represents the heart of a computer system, and it places two different types of requirements upon the auditor. First, as for the areas mentioned previously, a number of procedures need to be examined with respect to their impact on system security and integrity. These include the development and use of programming standards, the enforcement of documentation standards, the independent testing and examination of new software (systems, applications, modules, or simply patches to existing programs), and the independent operation of that software on the production computer system.

The separation of duties goes beyond the aforementioned split among development, test, and operation. The development of programs is (with today's technology) an inherently "buggy" process. Thus, there needs to be a separation of the production computer system from the development system, of the production files from the test data. Privacy considerations are likely to lead to the aforementioned test group being delegated the responsibility to provide programmers with "sanitized" test data, data that are like "live" data but that do not contain meaningful identification such as actual names or addresses.

The second type of responsibility for the auditor in this area concerns the program design process. Computer programming technology is such that a wide variety of functions can be included in a program at little cost if planned for in the original design. However, it is often prohibitively expensive to include any of the same functions after program development has been undertaken. Thus, it is critical that the auditor play a significant role in the program design process. It is at this point that he must seek to include the proper programmatic control features and to add the necessary audit trails to facilitate his auditing of the system when it becomes operational. The continued integration of functions within the computer system often leads to the loss of customary audit trails, but the technology is such that the auditor cannot afford to wait until he notices that the audit trails have vanished.

Personnel

Many aspects of this area are already familiar to auditors, including employee bonding and employee background investigations. However, surprisingly little is done to make the data processing employee realize that he is in as much a position of trust as is the employee of a bank trust department. A great deal of confidential or sensitive information passes through the computer system, and the employees frequently have access to it. Consideration must also be given to the protection, following an employee's termination of employment, of the program developments and sensitive data previously exposed to him.

Normally, considerations of employee relations and unionization are not particularly the province of the auditor. However, the vast concentration of organizational assets in computer systems and the critical role played by computers in the daily affairs of organizations, coupled with increasing attempts at unionization in the data processing field,[8] do have an impact on the assessment of system security, operability, and integrity.

Recovery

Disaster is an unpleasant subject, which is often the reason why human beings do not plan for it. Yet, disasters can and do happen. Thus, the auditor must be concerned with his organization's ability to recover and resume normal operations. The usual considerations include backup power supplies, spare equipment or excess capacity, off-site data storage, and backup computers (on-site, at other locations, or even in other organizations).

However, technology is making the assurance of adequate backup in the event of major disaster an increasingly difficult task. The development of large, centralized computer systems, particularly those with extensive telecommunication links, has resulted in systems that are expensive to replicate and in applications that are difficult to move from one site to another because of communications or other special requirements. This frequently results in a disaster plan that is simply a prayer that disaster will never strike.

THE COMPUTER AS AN AUDIT TOOL

The computer is not solely a villain in adding to the problems confronting the auditor; it also brings tools that can aid him in carrying out his responsibilities. Computer power has greatly enhanced the controls that can be employed in application programs, in monitoring system processing, and in controlling and logging facility access. Computer power has also greatly increased the types of tools available in conducting an audit. The following three types are illustrative.

Audit software

In the past several years many software packages have been developed for use in auditing computerized systems. These packages have primarily been intended to permit the auditor to access computer files independently with a minimum of effort, for the purpose of sampling, totaling, listing, comparing, and so on. Conceptually, these tools provide no functional change in the work of the auditor; he merely uses the computer to "read" the files that he can no longer read manually and to automate the computations on those files that for many years were performed manually. A very good description and comparison of the more commonly used audit software that is available may be found in Adams and Mullarkey.[9]

Dummy branch

The model division or dummy branch is a very powerful tool for the auditor that provides a conceptually different auditing approach. The auditor can introduce test transactions into the system through the dummy branch without having to back them out, without impacting regular company operations, and without introducing the additional control problems associated with audit personnel making direct changes to company records. The dummy branch also provides a mechanism for testing strings of previous transactions for a particular account, salesman, or branch to see whether the previously obtained results are replicated.

Despite the power of the dummy branch as an audit tool, it is not widely used today, largely owing to time and cost considerations. However, the path of technological development of computing may well remove or reduce this limitation (see *Audit Implications* below).

Contextual audits

Computer processing, coupled with more extensive computer-based data files, is permitting auditors to make contextual audits across time periods and operating division boundaries rather than having to examine a record or transaction in isolation, out of context. Thus, for example, an adjustment made to a payroll record falling within certain limitations will be accepted as legitimate, since errors are made and do have to be corrected. However, should that adjustment figure be repeated each pay period, there would be cause for further investigation. Similarly, the on-duty time of transportation operating personnel can be correlated with equipment dispatch records and with the independent records of other types of crew members serving on the same equipment. Thus, the auditor can bring to bear tools of much greater power in the conduct of his audit investigations.

AUDIT IMPACTS OF COMPUTING TECHNOLOGY CHANGES

The items discussed in the previous sections reflect developments that affect the work of the auditor today. However, the situation is not static. Computing technology is continuing to change rapidly, and the changes will have an impact on computing styles, which, in turn, will affect security and audit requirements. Therefore, it is important to consider some of the prospective changes and their derivative impacts.

These prospective changes are examined below in four steps. First, the changes in the technology used in various computer system components are examined. Then the impacts of these changes on the use of these components are explored. Next, the collective impacts of these changes on computer system design (hardware, operating systems, and applications software) are treated. Finally, the derivative implications of these system changes on the audit function are discussed.

Prospective technology changes

Recent advances in solid-state circuitry are rapidly being incorporated into currently manufactured computer systems. It is forecast that development trends in large-scale integrated (LSI) circuitry will continue, leading to memory and processor components that are smaller, faster, and cheaper. The use of LSI circuitry will also aid equipment reliability and maintainability. Not only does the use of LSI permit more fault detection and correction hardware to be included in the equipment design, but its packaging and reduced circuit costs permit circuit failures to be isolated and replaced at the chip level rather than at the circuit level. As a result, memory and processing equipment should not only be more powerful and cost effective but also more reliable in operation and more readily repaired when inoperable.

A variety of trends are forecast in the area of I/O and data transmission equipment. Magnetic storage media will probably retain their dominant role. However, storage density should continue to increase, and storage cost should drop. Access times and transfer rates should continue to improve. Printer technology is likely to advance significantly in the nonimpact area, providing much faster output rates, as well as graphic and image reproduction capability. The devlopment and use of programmable front-end communication processors is expected to continue, so communication processors may soon become an integral part of most standard systems. Digital communication circuits will become much more common, permitting not only reduced costs but also faster, more reliable communications.

Relative progress in the software area is not expected by many to be nearly as striking as that in the hardware area. Operating systems on medium- and large-scale computers are expected to grow larger and more complex as they are called on to provide additional services. In addition, more design emphasis will be placed on system integrity and security considerations. Thus, the processing power required to support the operating system should continue to increase. Program development will continue to constitute a large part of the total cost of computing. Although there are a number of changes on the horizon that should improve programmer effectiveness, these developments are not likely to have as significant an effect upon cost as hardware developments. Hence, programming should become relatively more expensive. The use of microprogramming capabilities for support purposes is expected to spread, permitting more effective use to be made of hardware for particular applications, as well as facilitating certain types of program development.

Impact on usage

The increased speed and reduced cost of circuitry should support the trend toward faster and more powerful processors within the main product lines of computer manufacturers. More applications should thus become feasible, either through reduced cost or through reduced processing time. This should assist the spread of on-line processing systems in many commercial areas.

However, these developments in circuit technology will also benefit minicomputer systems. More capable minicomputers will be able to perform many of the functions now thought to require a larger, general purpose computer. Capability coupled with the low cost of such minicomputers should facilitate the distribution of computing functions within an organization (see *Impact on System Design* below) and should permit greater specialization or dedication of equipment to particular applications.

The miniaturization of circuitry, in combination with low component costs, will probably lead to the incorporation of greater logical capabilities within equipment. This will apply not only to I/O devices and equipment control units but to terminals and user interface equipment as well. Thus, much more editing, formatting, and error correction could be handled at the terminal rather than at the computer. Microprocessors should continue to proliferate as a tool in support of additional equipment capabilities.

Technological developments should also result in faster memories to operate with the faster processors. In addition, forecast cost reductions will permit much larger standard memory sizes. Although larger memories will facilitate program development and will permit more efficient program execution, they will also support a continued need for complex multiprogrammed operating systems, with all the overhead and protective mechanisms associated with serving multiple users "simultaneously."

The availability of larger volumes of on-line random access storage at lower cost will probably facilitate greater use of on-line systems, as well as greater integration of an organization's files. Continued system integration implies a further loss of traditional audit trails, unless specific steps are taken to design audit trails and controls into individual applications.

Advances in I/O equipment should enhance many applications. More input should be possible via mechanical scanning without reliance on human keystroking. Image input as well as output should become possible, permitting, for example, an image of the pay-to-the-order-of line from a check to be input and subsequently printed on a customer statement. With the incorporation of increased logic in terminal devices, more input errors should be caught and corrected at the time of input, eliminating more costly error detection and correction in downstream processing.

Networking should be facilitated by the availability of digital communication links, as well as by the greater use of front-end processors. This should enable more accurate communication as well as more appropriate "human interfaces" to be developed for systems. Front-end processors will also facilitate the shifting of loads to other processors (e.g., in the event of equipment failure), permitting more reliable service to be offered with a greater service availability.

Operating systems are expected to offer additional services to programmers and users. This should help to reduce the growing cost of program development, but it is likely to do so at the cost of larger operating systems and increased use of processor cycles. However, the additional functions performed for the user are likely to be considered worth the cost, particularly in view of the expected decrease in computing costs.

Programming costs are expected to remain relatively high, even though many attempts will be made to reduce them. In addition to new operating system support features, further specialized software systems are likely to be developed to facilitate development and production usage in particular application areas. Use of data base management software should become more widespread, aiding organizations in dealing with increasing volumes of readily accessible files. Greater use of structured programming, chief programmer teams, and similar approaches is anticipated and should improve programming quality and cost effectiveness.

An interesting expected trend is the increasing use of hardware to aid software systems. Thus, specialized microprograms or "firmware" should be increasingly used to adapt computation to support certain types of processing. These tools should facilitate regular program development as well as program execution.

Impact on system design

The changes and developments discussed in the previous two sections are likely to support two opposing trends in the overall architecture or structure of computer systems: a trend toward central systems and an opposing trend toward distributed systems. Short of some unexpected developments, it is anticipated that both types of systems will proliferate, the choice depending on the particular circumstances confronting an organization.

Central systems

Economies of scale for both capital equipment and operational support, more powerful systems, and more capable software should facilitate the development of what might be termed central computer systems. Such systems are characterized by an aggregation of one or more large-scale processors or multiprocessors that are linked together at a single location. Most computing takes place at this central site, and most files are stored there, accessible to all the computers. Users throughout the organization can input data and receive outputs via terminals attached to a telecommunication network. The key distinction between this form of system organization and the distributed form (see below) is that processing capability and data files are stored at a single location in a star network rather than being distributed geographically and hierarchically throughout the network. It is interesting to note, however, that telecommunications can play a major role in either form of network.

The central type of system organization will favor an integrated file system. Given the cost and performance data mentioned above, the availability of all files to all the processors in one location should push an organization toward broader and more integrated file usage. In a similar manner, application processing steps should show increasing integration, making for more effective processing.

Centralized processing can have an advantage over distributed computing in that there need be no coordination of processing between hierarchical levels, no transfer of files between locations, and so forth. The central system is also better able to play a load-leveling role, since more averaging can take place on a large system than on a number of smaller systems. On the other hand, a central system of equivalent capacity is likely to run more heavily loaded, since a greater portion of its power must be used in support of the operating system.

Other impacts include the greater variety and sophistication of services that can be made available to users of such a system and the ability of a facility to attract more capable staff members. Currently, large powerful computer systems are viewed by computing professionals as being the more attractive systems on which to work. Historically, more new developments and more exciting activities have been associated with these systems. To the extent that this continues, the use of such "advanced" computing systems should be a positive factor in recruitment.

Distributed systems

The decreasing cost of hardware, coupled with enhanced capability, should facilitate the development of what might be termed distributed computer systems.[10,11] Such systems are characterized by a distribution of computational capability, both geographically and hierarchically. That is, certain types of processing may be done on small local computers, while other types of processing (or portions of processing steps) may be forwarded to more capable processing equipment at a higher level.

Distributed systems can follow a variety of patterns. Hierarchies may be defined by the relative computational power at each level or by the files or file contents available at each level. Individual processors may handle a variety of jobs or be dedicated to particular applications.

An additional impetus toward distributed computing may come from privacy and security considerations. Present-day operating systems are notoriously vulnerable to penetration. This poses a particular problem in connection with the central systems, for "all" users and "all" files are on the same system, with only the operating system preventing unauthorized access to files. The distributed design has the advantage of a simpler (and potentially more secure) operating system and of some physical barriers to file access. This may prove to be a powerful selling point in a world that is becoming more and more concerned about computer security and data file privacy.

A distributed system can have several impacts. The greater specialization of the use of each computer can reduce the complexity required in the operating system. Thus, less memory and storage space need be devoted to the operating system; fewer computer cycles will be needed to run it; less effort will be needed to develop it; and more security "holes" can be avoided.

The greater modularity that can be built into a distributed system can permit greater system flexibility. Growth can be accommodated by adding processors or by replacing processors incrementally. Failures have less of an impact on total performance, since a larger fraction of the total system remains operable. When many smaller processors are employed, it may be economically possible to switch in a replacement or standby processor to take over the function of the disabled one.

Distribution may or may not entail greater use of communication equipment, depending on the application. Processing performed locally reduces the need for communication to a central system. However, the passing of information and file data between hierarchical levels can entail greater communication requirements for certain designs.

Although moving data around a distributed network requires more emphasis on network security and control, such a network can lead to enhanced security. Efficient message switching and data flow requirements lead to greater standardization of requests, controls, and procedures for within-system communication. This structure, coupled with simpler system logic, can provide a positive increment to system security.

Audit implications

The prospect of the above developments has a number of implications for the auditor. These implications are discussed in three areas: those that relate to all systems, those that relate to central systems, and those that relate to distributed systems.

General

The continued decrease in the cost of computing should be beneficial to the auditor. Running tests or programs should become cheaper; running times should be reduced by the greater power of computing equipment. Larger samples should be possible without impacting ongoing operations more severely. Further, it may become possible to use tools never used previously in an organization because of their cost. The use of a dummy branch is a case in point.

The expected loss of audit trails will require greater involvement by auditors in the program development process. Without audit intervention, audit trails are likely to continue to be absorbed into more integrated processing steps, leaving fewer intermediate outputs. Further, with the probable continued high cost of software development and modification, it is imperative that appropriate audit trails be built into new software at the design stage rather than having to be retrofitted after development.

It is likely that many more security features, as well as monitoring tools and facilities, will be contained in computer operating systems. These have great potential for the auditor, but it will be necessary for him to learn how to use these facilities effectively and to incorporate their use into own auditing procedures.

Central system

The integration of data files should lead to greater use of data base management and report generation systems. Although these should ease data base access problems, their use will make more files readily accessible by more individuals. Use of these systems will also permit data from several files to be combined with much less effort, an action that could lead to more serious invasions of privacy.

The complex operating systems of large central systems will probably continue to have security weaknesses that can be exploited by possible penetrators. Thus, it will be more difficult for auditors to ensure the integrity of computer systems. Further, the greater number of programs and files on such a system should make the facility a more inviting target.

The proliferation of terminals at scattered locations, each having direct access to a system with so much information available, poses another problem. Greater attention will have to be paid to the physical access control for these terminals, as well as to the logical control over what can be accomplished by them.

Backing-up such a central system can pose problems because of the telecommunication interfaces at a single site and the large size of a "unit" of computing. Thus, the availability of a computer at another location will not suffice for backup purposes without a great deal of advance planning and (probable) contingency hardware expense.

Although processing capabilities and peripheral equipment capabilities are likely to reduce the audit trails available as a processing byproduct, the effect will be somewhat more pronounced in the central system. With less coordination between levels or nodes, there will be less need and opportunity for intermediate logs to be created.

Distributed systems

The frequently simpler logic of the distributed systems should make audits easier and provide greater assurance that processing is proper and that file integrity is being maintained. The organization of a distributed system provides both greater opportunity for audit monitoring and additional audit tools. In particular, the expected reduced cost of minicomputers, in conjunction with enhanced capabilities, should bring the development of an audit computer into the realm of feasibility. Such a computer could be integrated with the distributed system; yet it could enhance the desired isolation of audit programs and functions.

The possibility of more file movement and system-initiated transfers of programs and data within a distributed system should increase the importance of network design and management. Network protection will become more critical. Although all programming might be handled centrally and distributed to remote processors, a single individual with physical access to a processor at a remote location could load substitute programs and do considerable damage to the network.

Control can be a problem in distributed systems, since remote locations cannot always be staffed and operated at the same level or to the same standard as a central facility, owing to the smaller scale of the operation. Thus, more care may have to be exercised in this area, and increased audit attention may be required.

SUMMARY

The growing role of computing in many organizations, the evolution of computer files into significant assets, and the derivative impact of privacy considerations are leading to a much greater concern for computer security and computer system integrity assurance. These developments are impacting the auditor, adding to his computing-related responsibilities in the areas of physical security, access control, operating procedures, programming, personnel, and recovery. However, computer techniques have also provided additional control and audit tools to assist the auditor.

Computing technology is not static, and future changes in computing technology will affect the manner of computer use and hence the computer security facilities and requirements of the future. These changes will, in turn, impact the auditor, placing new demands (as well as providing new opportunities) for him to ensure the security and integrity of his organization's computing systems, data, and operations.

REFERENCES

1. Parker, D. B., S. Nycum, and S. S. Oura, *Computer Abuse,* Stanford Research Institute, Menlo Park, California, November 1973, 131 pp.
2. American Federation of Information Processing Societies, *System Reveiw Manual on Security,* AFIPS Press, Montvale, New Jersey, 1974, 109 pp.
3. U.S. Department of Commerce, National Bureau of Standards,

Guidelines for Automatic Data Processing Physical Security and Risk Management, Federal Information Processing Standards Publication 31, June 1974, 92 pp.

4. French, N. and E. Holmes,"Privacy Issue Stirring on, off the Hill," *Computerworld,* Vol. VIII, No. 43, pp. 1-2, 6, October 23, 1974.
5. Evans, A. Jr., W. Kantrowitz, and E. Weiss, "A User Authentication Scheme Not Requiring Secrecy in the Computer," *Comm. ACM,* Vol. 17, No. 8, pp. 437-442, August 1974.
6. Purdy, G. B., "A High Security Log-In Procedure," *Comm. ACM,* Vol. 17, No. 8, pp. 442-445, August 1974.
7. Friedman, T. D. and L. J. Hoffman, "Execution Time Requirements for Encipherment Programs," *Comm. ACM,* Vol. 17, No. 8, pp. 445-449, August 1974.
8. Schwab, B. and M. Thompson, "Unionism in Data Processing," *Datamation,* Vol. 20, No. 10, pp. 61-62, 64, 69, October 1974.
9. Adams, D. L. and J. F. Mullarkey, "A Survey of Audit Software," *J. Accountancy,* Vol. 134, No. 3, pp. 39-66, September 1972.
10. "In Your Future: Distributed Systems?," *EDP Analyzer,* Vol. 11, No. 8, pp. 1-13, August 1973.
11. Joseph, E. C., "Innovations in Heterogeneous and Homogeneous Distributed-Function Architectures," *Computer,* Vol. 7, No. 3, pp. 17-24, March 1974.

Area Director:
Susan H. Nycum
MacLeod, Fuller, Muir and Godwin
Los Altos, California

Legal aspects of computer management

The area "Legal Aspects" has particular appeal to computer supervisors and managers, facility directors, comptrollers, personnel and labor relations officers and others concerned with the management responsibilities of computer technology.

The sessions will focus on identifying legal problems that managers may encounter in their daily activities and discuss some of the approaches to them that have proved successful and the pitfalls to be avoided when a situation arises.

The first session, legal responsibilities in buying, using, selling data processing, will be chaired by Robert P. Bigelow, Boston attorney, who has been active in the computer law field for 20 years. Mr. Bigelow is a noted lecturer and author of numerous publications including *Computer Law and Tax Report,* the *Computer Law Service,* a multi-volumed treatise on computer law, and co-author of a forthcoming book, *Your Computer and the Law.* The session will deal with contracts, protection of proprietary rights and non-contractural responsibilities for computer use. Richard L. Bernacchi, partner in Irell & Manella, Los Angeles, and co-author of *Data Processing Contracts and the Law* will address contracting problems. Mr. Bigelow will discuss proprietary protection and Susan H. Nycum, member of MacLeod, Fuller, Muir and Godwin, Los Altos, participant in the NSF Computer Abuse Study and co-author of *Your Computer and the Law,* will discuss non-contractural responsibilities for computer usage.

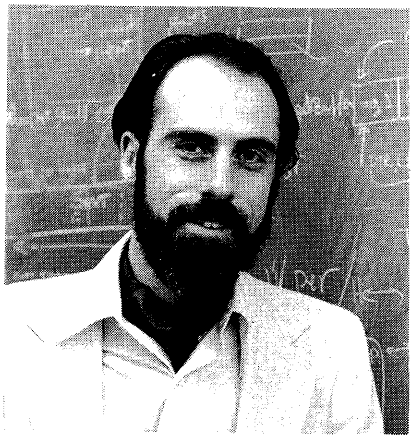
Sherwood Lewis, Assistant Corporate Counsel of Sanders Associates, Inc.,will chair the session on Antitrust and Regulatory Aspects. This session will examine antitrust and regulatory developments in data processing and telecommunications having immediate bearing on the data processing user and data processing operations. Emphasis will be placed on Government litigation and regulation, with comment on the effects of private actions, in both data processing and communications.

Mr. J. Thomas Franklin, partner in Sweeney and Franklin, Boston, will discuss Antitrust Activities in data processing. Mr. Lewis will examine regulatory and antitrust actions in telecommunications affecting data processing.

Roy N. Freed, Boston attorney, author of *Computers and Law, A Reference Work,* and national authority on computers and law is chairman and sole speaker of the session, "A Guide to Computer-Related Tax, Insurance, Recordkeeping and Labor Questions." Mr. Freed is noted for his practical approach to these complex and important issues facing EDP management.

The final session chaired by Ms. Nycum will bring the previous session chairmen together to view the future trends of the law and computer interface with an emphasis on international developments.

Each session will provide opportunities for interaction between the speakers and the audience.



Area Director:
Vinton Cerf
Stanford University
Stanford, California

International dialogue

Any Conference the size of NCC attracts specialists from all over the world. Indeed scattered in the various technical sessions are contributions from people in Japan, Canada, England, Germany, France, and Brazil. In addition to this individual participation, there is a set of four sessions focusing on topics that were found to be of special interest to non U.S. participants. They are standardization, especially the computer connection to communication networks, but also for input/output and software. We have arranged four sessions, with significant non-U.S. participation to articulate these areas.

The initial session (#44) provides a status report on public packet switching networks. In the last decade, packet switching has grown from an experimental idea to a commercially viable communication medium. The emergence of the value added carriers in the U.S. and of packet switched public nets in other countries, underlines this development. The panel will explore some of the technical, political, and economic issues facing these new carriers, as viewed by those who are supplying the service. Of special interest is the interconnection issue, particularly in an international context.

The next session (#50) turns to the policy questions that arise. The quality and reliability of existing telecommunication services will have to improve to meet the new needs, and policies concerning interconnection, sharing, tariffs, regulation, and balance of payments, to name a few, may have to change too. Even our institutional organizations may be affected by the need to cope with a vastly improved and more interconnected world wide telecommunications system. This panel will explore some of the short and long term policy issues which must be resolved before international data communication becomes as accessible as voice communication is today.

The third session (#56) turns to the economics of interface standards. Computer interface standards have been supported by many manufacturers and users as the means of reducing system cost and increasing flexibility. Others claim that standards will increase costs and damage the computer industry. This panel of independent U.S. manufacturers and users will discuss some problems and benefits of interface standardization.

The final session (#63) will extend the discussion of interface standards to the international arena by including panelists from Western Europe and Japan. They will also include some discussion on software standards to round out the day.

Determination and analysis of a standard interface model derived from a medium speed line printer

by GARY E. JONES
ODEC Computer Systems, Inc.
Warwick, Rhode Island

INTRODUCTION

The need for the development of a standard interface for computer associated equipment is a subject that requires investigation based on analytical facts rather than emotion. To establish a baseline for the definition of an interface model, a medium speed line printer utilizing 22 interfaces for a given printer model provides adequate interfaces for investigation. Such a model has many similarities relative to any interface consideration. All data presented is factual and represents an actual functioning interface to a controller. Most interfaces are associated with terminals, minicomputer or medium scale computers.

ANALYSIS OF INTERFACE DIFFERENCES FOR A MEDIUM SPEED LINE PRINTER

A consideration of the differences between all interfaces makes it possible to analytically investigate the many parameters present. Such a parameter study gives insight to the significance of a standard interface.

Hardware requirements and limits

The connectors and cable requirements for the interface circuitry vary greatly between controller manufacturers. Changes in connector configurations cause cabinet modifications, extra paper work and manufacturing configuration control. Such necessary complications add cost to the product to satisfy the varied customer requirements.

One variation associated with the connector is pin assignments. Out of the 22 interfaces being considered, 16 different pin configurations exist. The number of pins contained in the various connectors is a low of 22 with three coax, and a high of 50 pins. Of the actual pins used, a low of 20 and a high of 32 was present.

Three physical size differences of connectors occurred. A further complication was the use of both alpha and numeric pin designations. Connector variability causes extra definition and paper work requirements.

Functional definitions and variations

The interchange of data between the printer and controller may be basically defined to consist of a minimum number of control functions. Table I lists general printer control functions and the occurrence data. It can be seen that only the data strobe and basic data line signals are present in all interfaces. Such signal variations cause configuration differences and extra paper work requirements.

Table II lists the total functions utilized. No one interface utilizes all of the functions listed in Table II. The various functions defined for each given interface require design changes from minor modifications to entire printed circuit board design.

Options

Certain of the functions included in Table II are considered to be optional requirements. Such requirements add speed advantages or economic impact and should be considered as a variable even in standard interface definitions.

The VFU (Vertical Format Unit) variation between teletype and IBM type paper tape is not an optional requirement. However, the difference between a two channel unit and a 12 channel unit requires considerable circuit design differences. Extra cost is associated with the 12 channel unit and the interface requirement for 12 channel is the most complex. The Vertical Format Unit is used for programmed control of paper movement.

Different character fonts allow speed advantages for given data format requirements. This difference in function is significant enough to be included in option definition.

Interface signal title variations

Table III lists the total interface signal titles utilized. Variations in signal titles with respect to controller requirements causes cable wire run list differences and schematic

TABLE I—General Printer Control Functions

FUNCTION	*STANDARD/VARIES*
Alarm Condition	Varies
Character Ready	Varies
Data Lines 1–6	Varies
Data Lines 1–7	Varies
Data Strobe	Standard
Line Ready	Varies
On Line	Varies

changes. Table IV lists the signal titles that are associated with the interface defined to be standard by the printer manufacturer. The standard interface has been utilized with over 75 unique customer's controllers. The 21 other interfaces considered include various combinations of connector differences, functional differences, and signal title differences. The various levels of change cause paper work differences. Table V lists the interface model parameters with respect to the interfaces considered. It is evident that a standard interface definition would reduce costs by elimination of much of the workload as defined by Table II and Table V. A standard interface utilizing signals definition similar to Table IV and a definition of cable and functions would greatly reduce the need for the many configuration differences required.

ANALYSIS OF REASONS FOR INTERFACE DIFFERENCES

The reason for the many different ways of doing a similar function is directly related to the many different printers available. Each printer manufacturer selects unique functions and cable requirements. Other peripheral manufacturers respond likewise. A system manufacturer is then faced with connecting a number of peripheral units to a controller. Once the selections are made, a cable and connection requirement emerges. As competition grows, the system manufacturer becomes interested in attempting to mutilize other peripherals, mostly due to either economic or reliability considerations. Figure 1 shows four printers connected to four controllers illustrating the requirements that may be imposed on a printer manufacturer to be competitive. Referring to Figure 1, the following relationships are determined.

Printer A has three interfaces with interface A1 considered to be a standard interface of the printer manufacturer. The standard interface for Printer A is compatible with controller 1 and controller 2. Interfaces A2 and A3 are compatible with controllers 3 and 4 respectively.

TABLE II—Total Functions Utilized

Automatic Print	
Automatic Print and Paper Advance	
Differential Interface—Line Driver/Receiver	*
Feed After Print	
Feed Before Print	
Motor Off Delay	
Negative Logic	
Paper Jam Detection	
Paper Runaway	
Paper Tear Detection	
Positive Logic	
Print on Command	
Serial/Parallel Interface	
Special Termination Resistors	
Special Test Character	
2 Channel VFU (IBM/TELETYPE)	*
12 Channel VFU (IBM/TELETYPE)	*
48 Character Set	*
64 Character Set	*
96 Character Set	*

* OPTIONAL

TABLE III—Total Interface Signal Titles

$\overline{\text{ACK}}$
Alarm
$\overline{\text{Alarm}}$
Alarm Serial
$\overline{\text{Belt Mtr in Speed}}$
Buffer Rdy
Buffer Ready
$\overline{\text{Buffer Ready Serial}}$
Char Rdy
Character Ready
Data Bit 1–6
Data Bit 1–7
$\overline{\text{Data Bit 1–7}}$
Data Strobe
$\overline{\text{Data Strobe}}$
DSTB
EOPL
VFU Channel 1
VFU Channel 2
VFU Channel 8
Gnd
$\overline{\text{Master Reset}}$
$\overline{\text{MA-STAT}}$
On Line
$\overline{\text{On Line}}$
Paper Out
PE
Print
$\overline{\text{Ready}}$
$\overline{\text{Select}}$
Serial Data
+5V

TABLE IV—Standard Interface Signal Definition

$\overline{\text{Alarm}}$
$\overline{\text{Belt Motor in Speed}}$
Buffer Ready
Data Bit 1–7
Data Strobe
Line Ready
$\overline{\text{Master Reset}}$
On Line
Paper Out
Ground
+5V

Printer B utilizes four unique interfaces to be compatible with the four controllers.

Printer C1 standard interface is compatible with controller 3 and controller 4. Interfaces C2 and C3 are compatible with controllers 1 and 2 respectively.

Printer D1 standard interface is compatible with controller 1 and controller 2. Interfaces D2 and D3 are compatible with controller 4 and controller 3 respectively.

Figure 1 shows that a printer standard interface as defined by the printer manufacturer is not compatible with other printer standard interfaces, and therefore not compatible with all controllers.

It is also shown in Figure 1 that each controller accepts four different printers identically or demands a standard interface. In order for all printers to be compatible to all controllers, a total of 13 interfaces is required. A standard interface theoretically could reduce the number to 1 interface. Typically, many independent printer manufacturers are required to design many different interfaces to be competitive. It is not uncommon for controller manufacturers to also design many different interfaces to be compatible with printers and other devices. In order that controllers second source devices without change to the system, many interface circuit designs are required.

Printer standard interfaces

Most independent printer manufacturers establish a standard interface. The standard interface is compatible with a certain number of controllers. In all cases where the standard interface is incompatible, unique interfaces occur. Considering a sample of customers that included the 22 unique interfaces, 75 customers utilized the standard interface described in Table IV. This would mean that competitive printer manufacturers would have to modify their interface to become a second source. Of all printer manufacturers, it is unlikely that any standard printer interface of one manufacturer would be compatible with any other manufacturer's standard printer interface. This fact insures that a lot of design will be necessary to insure compatibility to a number of controllers.

TABLE V—Interface Model Parameters

DRAWING	NO. OF VARIATIONS
Schematics	14
WRL	12
50 Pin Connector	1
37 Pin Connector	1
22 Pin—3 Coax Connector	1
Alpha Pin Designation	1
Numeric Pin Designation	1
Physical Configuration—Connector	3
PWA Assembly Drawings	15
Interface Cable	12
PWA—Artwork	11
Mini-Computer Additional Interfaces	5

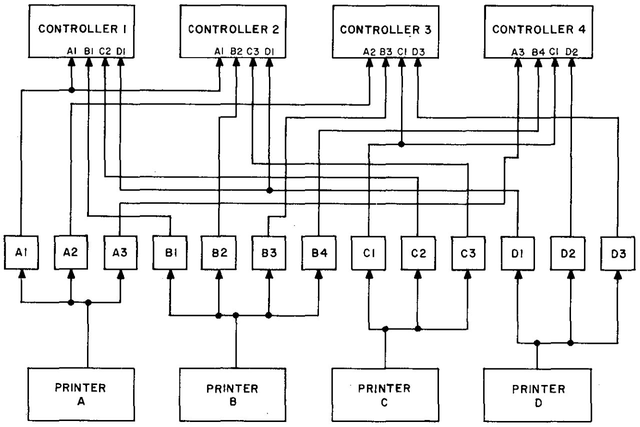


Figure 1—Controller to printer interface

Printer mini-computer interfaces

Many times two different interfaces are required to be compatible with mini-computer interfaces. In such cases, the printer standard interface is fed into a number of unique interfaces to be compatible with the many minicomputer types now available. Due to the lack of a standard mini-computer interface specification, at least five different external interfaces (for a given medium speed line printer) are required to be compatible with all minicomputer types.

SUMMARY

By analyzing the compatibility requirements for a single line printer, it can be seen that a great many varieties of interfaces exist. Most interfaces are unique to certain customers and are a second source to other printer manufacturers. In order to have a practical interface definition, operational definitions must be considered. Once the software programs for a controller are established, the printer must be designed to be compatible. Therefore, functional operations must be standardized along with connector definition and printer functions. In order for a printer standard to be established, a standard for controllers must also be established. In fact, a controller standard should be established prior to a printer standard consideration. Since each printer manufacturer has established a standard interface, not compatible for all controllers, it is evident that if a controller standard interface existed, the standard printer interface would be compatible. At the very least, a specification for standard interfaces would allow efficient design due to the evaluation of all parameters. At present, new conditions occur with such frequency, that total stabilization of interface designs is highly unlikely. As long as a manufacturer is a first source, the interest in a standard interface would be low. However, if the same manufacturer had interest in becoming a second source, the interest now may be high.

Second sourcing of components met with initial resistance from component manufacturers. However, due to economic pressures from users, second sourcing of components is now very common. In order to second source components, a standard specification for the component was generated. The same basic situation applies to computer systems. It is true that the definition of the standard is more complex; but a definition can be accomplished. As long as the user and the manufacturer are willing to change design and configuration, the pressure to define a standard interface will be lacking. The most important fact associated with the standard interface question is that extra cost is required if a standard is not generated. The cost of the extra design is, in most cases, totally unnecessary if a standard is established. When the user and manufacturer understand the unnecessary waste and cost, pressure will be applied to establish a standard interface. A standard interface specification would be as practical to a printer manufacturer as a component specification would be to a component manufacturer.

nfais/FID world inventory of abstracting and indexing services

by TONI CARBO BEARMAN

National Federation of Abstracting and Indexing Services
Philadelphia, Pennsylvania

BACKGROUND

The National Federation of Abstracting and Indexing Services (*nfais*) is a Federation of not-for-profit organizations and government agencies that are engaged in abstracting and indexing. The Federation was incorporated in 1958 under the name of National Federation of Science Abstracting and Indexing Services; the word Science was dropped from the name in 1972, thus broadening the membership to include services in the humanities and the social sciences. There are currently more than thirty member services. There is no individual membership in the Federation.

The goals of the Federation are to assist the member organizations in their efforts to improve their services and operations; to act as a communication forum for members; to undertake specific projects on behalf of members that no one single member organization would undertake alone and that would be clearly useful to the majority of member organizations; and to act as a national spokesman for the collective member organizations.

The Federation Internationale de Documentation (FID) is an international organization consisting of national members drawn from many countries of the world. The organization became federated in 1924. The aim of FID is to promote research and development of documentation through international cooperation in the fields of science and technology, social sciences, arts and the humanities.

Both Federations have published directories of abstracting and indexing services. This paper describes an international project undertaken by these organizations to develop a world inventory of abstracting and indexing services. This project will merge the previously published information, add information on additional services, develop a machine readable record of the information, and provide periodic updates to the file.

Since abstracting and indexing services of the world constitute secondary information sources from which can be obtained important information about primary publications in all subject areas and in many different languages, knowledge of these services is essential to everyone interested in keeping up with the world literature in any subject area.

INTRODUCTION

The purposes of the joint *nfais*/FID project are:

1. to provide a machine readable inventory file of information on the world's abstracting and indexing services with the capability of cross indexing and sorting the file to provide printed publications and other specialized services;
2. to provide a guide to abstracting and indexing services as a machine readable data base in a format that is capable of being searched by subject, country, language, and other characteristics;
3. to provide a detailed, authoritative world-wide printed directory of abstracting and indexing services which can be used by librarians, information specialists, and researchers as a bibliographic tool.[1]

Work on the Joint Project commenced in mid-1971 with the FID effort funded by UNESCO UNISIST and the *nfais* effort funded by the National Science Foundation Office of Science Information Service under GN-28849.

OUTLINE OF *nfais*/FID PROJECT

1. To merge and update the data already contained in two guides published by FID and *nfais*.
2. To collect current data on indexing services in science, technology, the social sciences and the humanities.
3. To compile the data in machine readable form and develop software that provides for periodic input updating and searching of the file.
4. To publish a revised printed directory of abstracting and indexing services.
5. To develop promotional materials to describe the availability of the special services that might be generated from the mechanized data base.
6. To establish procedures to update and maintain the data base after the funded project is completed.[1]

PROGRESS TO DATE

Criteria and Size of File

A set of explicit criteria and a series of guidelines were developed that establish standards for inclusion of a service in the inventory by Jane Collins of the Library of Congress, Science and Technology Division, who has been working as a consultant on this project. These refer to such characteristics of services as number of items included each year, whether the service is still being published, how frequently the service is published, what other services are available in that specific subject area, etc. Many services which were included in one of the published guides were rejected for inclusion in the inventory because they failed to meet one or more of the criteria. Early estimates of size of the file have been proven to have been too large. The file contains approximately 2,500 records.

DATA ELEMENT DEFINITIONS

The data elements to be included in the Inventory have been identified and defined. A list of 101 data elements grouped into 11 main classes for the printed services and an additional seventy elements in three additional classes for the machine readable services define the file (copy attached). As far as possible, existing codes were used to identify specific elements. For example, the American Society for Testing and Materials' (ASTM) journal Coden has been used for the identification of journal titles (with provision for the later addition of the International Serial Number, ISSN) and the codes used by the Library of Congress MARC system to identify country of publication and language have been used.

COMPUTER BASED SERVICES

In developing the criteria for inclusion of services in the Inventory, it was agreed that services produced in forms other than printed (e.g., microfilm, computer tape, etc.) should be included. The previously published *Guides* had limited coverage to printed publications, and specified publications issued in other forms in the supplementary notes. In developing the plans for the Inventory, it was agreed that all services with a uniquely identifiable title should receive a separate main entry in the Inventory and be identified with its corresponding printed service, if one exists, by cross references. At this time, there are some 200 services in machine readable form, approximately 90 of these are sections of either *Excerpta Medica* or the French P.A.S.C.A.L. system.

The expanded data element list mentioned above is applied in describing the machine readable services. The additional elements for these services were based on those used in the American Society for Information Science Special Interest Group on SDI Survey published by the American Institute of Physics[2] the directory prepared by the FID/TM Committee and published by the CSIR Library in South Africa[3] and the directory published by the OECD.[4]

INPUT

A special coding sheet, called a Q/IS form (for "questionnaire/input sheet") was developed, and supplementary sheets for the machine readable services were designed. The records were input at FID using a Dura paper tape machine, which was later replaced by a Forester device. The paper tape has been converted to magnetic tape by the subcontractor, Illinois Institute of Technology Research Institute, which has also developed all the software needed for building and searching the file.

LIBRARY OF CONGRESS PARTICIPATION

In 1972, *nfais* negotiated with the Library of Congress to obtain assistance from a Library of Congress staff member on a contract basis to assist with the project. This arrangement provided for checking the Q/IS forms for accuracy; resolving questions generated at FID and *nfais* by reference to the source document or using bibliographic resources; and locating services to be input into the file.

THE MACHINE READABLE INVENTORY

The Inventory is expected to contain information on approximately 2,500 abstracting and indexing services, approximately 200 of which are in machine readable form. There will be approximately 500 additional entries which will be cross referenced from one title (for example, a former title or an alternate title) to the main entry. The data will include specific information on the following general areas: (1) title, (2) publisher, (3) history, (4) frequency, (5) price, (6) contents, (7) indexes, (8) language, (9) media, (10) complementary services, and (11) subject. In addition, for the machine readable services, data on the following will be included: (1) search elements, (2) data elements present, and (3) tape specifications.

The file will be maintained on magnetic tape which will be available for searching by bona fide researchers for a fee. The details of this have not yet been worked out, but it is expected that the file will be a valuable research tool because of the comprehensiveness of the project and the search capabilities built into the file. A person interested in identifying those services in English and French, for example, issued since 1970 on the subject of Toxicology, which cover serials literature and patents should be able to have the file searched and receive a list of services specifically on his topic.

In keeping with its policy of supporting the use of extant standards, *nfais* has specified that the tape be available in accordance with the American National Standards Institute (ANSI) Z39.2-1971. Physically, the tape will be in ASCII code (see X3.4-1968), on 9-track 800 bpi density (see ANSI X3.22-1967) and labelled in accordance with ANSI X3.27-1969.

The printed directory is expected to be available January 1976. The printed volume will have detailed information on all the services included in the Inventory and will also include various indexes to the entries, such as subject, country, UDC number and publisher. Only the most up-to-date information on each service will be included in the printed directory. Before publication of the directory, complete information contained in the file for each service will have been sent to each publisher asking for corrections and updating. Based on these responses, the information will be revised so that all data in the printed directory will be the best information obtainable as of 1975.

UPDATING THE DIRECTORY

Procedures are currently being worked out to provide updates to the master file on a regular basis, so that the machine readable inventory will be periodically updated. The mechanism for updating the printed volume has not yet been determined. One possible method is the use of the FID and *nfais* newsletters to provide monthly and bimonthly updates, respectively, for the printed directories. This listing in the newsletters would be for partial information only. The complete information would be added to the machine file.

QUESTIONS AND SUGGESTIONS

The author would welcome information on new abstracting and indexing services for consideration for inclusion in the Inventory. Also, queries about the project and requests for announcements of the final products are welcome.

DATA ELEMENT DEFINITIONS FOR MACHINE-READABLE SERVICES

010—310 Same as for printed services.

350 PRICE ELEMENTS

351 Available for purchase
Indicate Y if yes. Specify: Not available for purchase. Include limitations, e.g., available only to non-profit institutions.
FIELD *Limited*

352 Available for lease
Indicate Y if yes. Specify: Not available for lease. Include limitations as in 351.
FIELD *Limited*

353 Current files price
Price of service's current files in domestic currency . . . Same as 370 for printed services. Specify any restrictions.
FIELD *Limited*

354 Price for back files
Price of service's back files in domestic currency in country of publication. Year is always to be included. Use special table for currency. See note above in 353. Specify any restrictions.
FIELD *Limited*

355 Price for sample files
Price of sample files in domestic currency in country of publication, year is always to be included. Use special table for currency. See note above in 353. Specify any restrictions.
FIELD *Limited*

356 Royalty fees
Price of royalty fees in domestic currency in country of publication, year is always to be included. Use special table for currency. See note above in 353. Specify any restrictions.
FIELD *Limited*

357 Price differentiation for type of use or type of user
Specify if there is any price differentiation for type of use or type of user (e.g., not-for-profit or for profit institutions or royalty fees charged for external use. If none, write none. List the prices, themselves, in 353-356 above.
FIELD *Limited*

358 Other
Specify other prices as in 353 above.
FIELD *Limited*

390 Price Notes:
Additional information concerning price not identified as a specific element.
FIELD *Open*

400 CONTENT ELEMENTS

Same as for printed services. Note that 440 information will probably have to come directly from the publisher.

500 INDEX ELEMENTS

Same as for printed services.

500 LANGUAGE ELEMENTS

Same as for printed services.

*600 SEARCH ELEMENTS**

601 Enriched titles containing added terms
Enriched titles containing added terms. Give average number of added terms per title, if known, otherwise indicate Y-yes.
FIELD *Fixed (3)*

602 Average number uncontrolled keywords selected by indexers per document.
Give number if known, otherwise indicate Y-yes.
FIELD *Fixed (3)*

603 Average number descriptors from a controlled thesaurus per document. Give number if known, otherwise indicate Y-yes.
FIELD *Fixed (3)*

604 Number, subject headings/document
Average total number of subject headings per document. Give number if known, otherwise indicate Y-yes.
FIELD *Fixed (4)*

605 Number classification scheme categories/document
Average number of categories from classification scheme per document. Give number if known, otherwise indicate Y-yes.
FIELD *Fixed (3)*

606 Classification scheme

* From published guide or information provided by publisher.

Name of classification scheme used, e.g., LC or UDC.
FIELD *Limited*

607 Source for subject headings, descriptors, key words
Full title of source or sources used for subject headings, descriptors or key words. Include publisher, city and year as in 005.
FIELD *Limited*

608 Average number of short phrases or word strings describing content per document
Give number if known, otherwise indicate Y-yes.
FIELD *Limited*

609 Other subject heading search elements
Specify.
FIELD *Open*

OTHER DATA ELEMENTS

610 Personal author
Personal author, first author only. Indicate Y-yes, if first author is present.
FIELD *Fixed (1)*

611 Additional authors
Personal authors in addition to first authors. Indicate Y-yes if present.
FIELD *Limited*

612 Corporate author
Indicate Y-yes if corporate author (e.g., society, institution or conference as document author) is present.
FIELD *Limited*

613 Author affiliation
Indicate Y-yes if the name of the institution or society with which the author is affiliated is present.
FIELD *Limited*

614 Author address
Indicate Y-yes if the address of the personal author is present.
FIELD *Limited*

615 Corporate author address
Indicate Y-yes if the address of the corporate author is present.
FIELD *Limited*

616 Document title
Indicate Y-yes if the title of the document (e.g., title of the article, book or report) is searchable.
FIELD *Limited*

617 Document language
Indicate Y-yes if the language of the document is searchable.
FIELD *Limited*

618 Document date(s)
Indicate Y-yes if the date(s) of the document is (are) searchable.
FIELD *Limited*

619 Country of publication
Indicate Y-yes if the date(s) of the document is (are) searchable.
FIELD *Limited*

620 Serial title
Indicate Y-yes if the date(s) of the document is (are) searchable.
FIELD *Limited*

621 CODEN
Indicate Y-yes if the date(s) of the document is (are) searchable.
FIELD *Limited*

622 ISSN
Indicate Y-yes if the date(s) of the document is (are) searchable.
FIELD *Limited*

623 Bibliographic reference (volume, issue, page of a given document)
Indicate Y-yes if the date(s) of the document is (are) searchable.
FIELD *Limited*

624 Accession number
Indicate Y-yes if the date(s) of the document is (are) searchable.
FIELD *Limited*

625 Report number
Indicate Y-yes if the date(s) of the document is (are) searchable.
FIELD *Limited*

626 Patent number
Indicate Y-yes if the date(s) of the document is (are) searchable.
FIELD *Limited*

627 Title of cited printed service
Indicate Y-yes if the title of the printed service corresponding to the tape service and cited by the tape service is searchable.
FIELD *Limited*

628 Issue of cited printed services
Indicate Y-yes if the issue (i.e., issue number or month of issue) of the printed service corresponding to the tape service and cited by the tape service is searchable.
FIELD *Limited*

629 Year of cited printed service

630 Citation of printed service
Indicate Y-yes if the citation (i.e., volume number and pages) of the printed service corresponding to the tape service and cited by the tape service is searchable.
FIELD *Limited*

631 through 633 are to be used for those services (e.g., Good Res. Repts.) which are formed from the combination of other abstracting/indexing services. The term, "source services" refers to those services (e.g., *S.T.A.R.* and *N.S.A.*) which are combined to form the service.

631 Source service journal title(s)
Indicate Y-yes if the title(s) of the source service(s) is (are) searchable.
FIELD *Limited*

632 Source service journal year
Indicate Y-yes if the year of the source service journal is searchable.
FIELD *Limited*

633 Source service journal citation
Indicate Y-yes if the citation (i.e., volume number, issue number and pages) of the source service journal is searchable.
FIELD *Fixed (1)*

634 Other searchable data elements
FIELD *Open*

635 NOTES: Additional information concerning the other searchable data elements not identified in a specific element.
FIELD *Open*

636 Abstract
FIELD *Fixed (1)*

650 *MEDIA ELEMENTS*

670 Machine readable tapes
Indicate Y-yes if service is available on tape.
FIELD *Fixed (1)*

680 Other
Specify in what machine readable medium the service is available other than tape (e.g., disk).
FIELD *Open*

685 DO NOT USE

690 Media notes
Additional information concerning the form of the service not identified as a specific element.
FIELD *Open*

700 *COMPLEMENTARY SERVICE ELEMENTS*

Search programs and SDI programs

770 SDI programs
Indicate Y-yes if SDI programs are available.
FIELD *Limited*

771 Retrospective search programs
Indicate Y-yes if retrospective search programs are available.
FIELD *Limited*

772 Source for programs
Give the name and address of the person who is the source for programs. Follow the instructions in 146 for the name and the instructions in 120 for the address.
FIELD *Open*

773 Country code
ISO TC46 Draft Standard, October, 1972. This code should be identical with the one used by the U.S. National Center for ISDS.
FIELD *Fixed (3)*

774 Programming language for search programs
Indicate the programming language(s) used for the search programs as listed in the publisher's brochures or obtain this information from the publisher.
FIELD *Open*

775 Hardware configurations for search programs
List the hardware configuration required as listed in the publisher's brochures or obtain this information from the publisher.
FIELD *Open*

776 Cost or range of costs for search programs
Give the cost or range of cost for the search programs (both SDI and retrospective available) in U.S. dollars.
FIELD *Open*

777 SEARCH PROGRAMS AND SDI NOTES
Additional information concerning the search programs and SDI programs not identified as a specific element.
FIELD *Open*

778 NOTES
Additional information concerning complementary service not identified as a specific element.
FIELD *Open*

800 SUBJECT ELEMENTS
Same as printed service.

830 Subject Notes
Additional information concerning subject not identified as a specific element.
FIELD *Open*

901 Code
Indicate the name of the code or codes used for the tapes.
FIELD *Limited*

902 Density
Indicate the tape density in bytes per inch.
FIELD *Limited*

903 Magnetic tape width
For magnetic tape *only*, indicate the width of the tape in inches.
FIELD *Limited*

904 Number of tracks (magnetic tape)
For magnetic tape *only*, indicate the number of tracks.
FIELD *Limited*

905 Parity (magnetic tape)
For magnetic tape *only*, indicate the parity.
FIELD *Limited*

906 Number of tracks (paper tape)
For paper tape *only*, indicate the number of tracks.
FIELD *Limited*

907 Number of columns (punched cards)
For punched cards *only*, indicate the number of columns.

908 Labels present
Indicate 1 if labels are present, zero if they are not.
FIELD *Fixed*

909 Label standard
Indicate the standard used for the label, if one is used.
FIELD *Limited*

912 Blocked/unblocked
Indicate if the tapes are blocked or unblocked. If they are blocked give the length in bytes (give maximum, if varying). Give the length of a byte in bits.
FIELD

913 Reel size
Indicate the size of the diameter of a reel of tape in inches.
FIELD *Fixed (2)*

914 Reel length
Indicate the length of a reel of tape in feet.
FIELD *Fixed (3)*

915 Character set
Indicate the number of unique characters. Indicate whether upper case only or both upper and lower case characters are present. Specify any other information concerning the character set.
FIELD *Open*

916 Required auxiliary files
Specify any auxiliary files (e.g., dictionary tapes, cross reference tapes) that are *required* for the use of the service.
FIELD *Open*

917 Optional auxiliary files
List any optional auxiliary files which are available.
FIELD *Open*

918 Documentation source
Give the name and address of the person who is the source for documentation for the service. Follow the instructions in 146 for the name and the instructions in 120 for the address.
FIELD *Open*

919 Country code
ISO TC46 Draft Standard. October, 1972. This code should be identical with the one used by the U.S. National Center for ISDS.
FIELD *Fixed (3)*

920 TAPE SPECIFICATIONS NOTES
Additional information concerning tape specifications not identified as a specific element.
FIELD *Open*

REFERENCES

1. Keenan, S. and M. Elliott, "World Inventory of Abstracting and Indexing Services," *Special Libraries* 64:(No. 3) pp. 145-150, March 1973.
2. Survey of Scientific-Technical Tape Services. Compiled and edited by Kenneth D. Carroll. New York, American Institute of Physics, 1970 (AIP ID 70-3; ASIS SIG/SDI 2).
3. Computer Tape Services in Science and Technology. Compiled by CSIR Library. Pretoria, South Africa, Council for Scientific and Industrial Research, 1970 (For FID/TM Committee).
4. Inventory of Major Information Systems and Services in Science and Technology. Paris, OECD, 1971.

BIBLIOGRAPHY

Abstracting Services: Volume 1, Science, Technology, Medicine, Agriculture; Volume 2, Social Sciences and Humanities. Federation Internationale de Documentation, The Hague, Netherlands 1969.

Abstracting Services in Science, Technology, Medicine, Agriculture, Social Sciences, Humanities. Federation Internationale de Documentation, The Hague, Netherlands 1965.

Computer Tape Services in Science and Technology. Compiled by CSIR Library. Pretoria, South Africa, Council for Scientific and Industrial Research, 1970 (For FID/TM Committee).

A Guide to U. S. Indexing and Abstracting Services in Science and Technology. National Federation of Science Abstracting and Indexing Services, Washington, D. C. 1960 (out of print).

A Guide to the World's Abstracting and Indexing Services in Science and Technology (Report No. 102). National Federation of Science Abstracting and Indexing Services, Washington, D. C. 1963 (out of print).

Inventory of Major Information Systems and Services in Science and Technology. Paris, OECD, 1971.

Survey of Scientific-Technical Tape Services. Compiled and edited by Kenneth D. Carroll. New York, American Institute of Physics, 1970 (AIP ID 70-3; ASIS SIG/SDI 2).

UNISIST; Study Report on the Feasibility of a World Science Information System. United Nations Educational Scientific and Cultural Organization and the International Council of Scientific Unions. Paris, France 1971.

Computing applied to societal problems

CHAIRMAN—DONALD L. THOMSEN, JR.
SIAM Institute for Mathematics and Society

OVERVIEW—Donald L. Thomsen, Jr.

The panel will present problems from three societal fields where computing has made significant contributions. The first field is cultural evolution, which is concerned with how both biological characteristics and new ideas spread throughout a society. Next, from that aspect of the environmental field which is concerned with waste, computer applications to ocean disposal will be discussed. The final presentation will concern programs for allocation of fire companies; this will be followed immediately by a computer demonstration at which time those attending may themselves operate the computer program. Throughout the session the panel will stress the usefulness and applicability of their investigations to the needs of society.

The Interaction Between Biological and Cultural Processes (Abstract of presentation)—Marcus Feldman and Luca L. Cavalli-Sforza

Most studies of evolution concern the genetic system and evolution occurs through changes in gene frequencies in populations. The study of the dynamics of phenotypes which can be specified, for example, at a cultural level has only recently been initiated. In the present work, we present a theory for both genotypes and culturally determined phenotypes. The individuals in the population have a cultural character and a defined genetic type. Both are transmitted by the parents, or members of the group to which the parents belong. The cultural character can be under natural selection. It is shown that selection on the cultural character can cause evolution at the biological level, as in the change of gene frequencies. The studies were based on new applications of classical theories of recursion systems as well as on numerically based computer work.

Computer Applications to Ocean Disposal Research and Engineering (Abstract of presentation)—Robert C. Y. Koh

The role played by computers in three areas related to the technology of disposal of wastes (either waste water or waste heat) to the ocean is discussed. First, the computer is necessary in the development and implementation of mathematical models for the purpose of predicting the effects of discharge on the environment. Unfortunately, this is limited at present primarily by a lack of adequate understanding in the basic processes involved making most models of only limited usefulness. To enhance this understanding, laboratory investigations are being undertaken, Thus the second area where computers are used is in laboratory data analysis sometimes on a real time basis. The third area where computers are used is in the analysis and interpretation of field monitoring data. This is important in assessing the impact, in establishing the background levels and the adequacy of various water quality standards. Examples of applications to actual systems will be discussed.

Programs for Allocation of Fire Companies (Abstract of presentation)—Jan M. Chaiken

Municipal fire departments routinely make long-term planning decisions concerning the location of fire stations. Capital expenses related to these decisions are ordinarily not large compared to the department's budget, but the manning costs can amount to $200,000 per unit per year. Thus the objectives in planning station locations include minimizing the total number of units while providing certain minimum levels of coverage for all parts of the city. In addition, the average response time to fires in the city, and the equity of response time among different areas of the city, are relevant criteria.

The New York City-Rand Institute has designed and tested several computer programs that assist municipal agencies in planning fire station locations. The principles on which they operate, data requirements, and output will be described. The programs will be available for operation by the audience at the conclusion of this session.

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES, INC. (AFIPS)

OFFICERS AND BOARD OF DIRECTORS

President

Mr. George Glaser
225 Warren Road
San Mateo, California 94402

Vice President

Mr. Paul W. Berthiaume
Electronic Associates, Inc.
185 Monmouth Park Highway
West Long Branch, N.J. 07764

Secretary

Dr. Dick Simmons
Data Processing Center
Texas A & M
College Station, Texas 77843

Treasurer

Mr. Marvin W. Ehlers
Square D Company
Executive Plaza
Park Ridge, Illinois 60068

Executive Director

Dr. Robert W. Rector
AFIPS
210 Summit Avenue
Montvale, New Jersey 07645

ACM DIRECTORS

Miss Jean Sammet
IBM Corporation
545 Technology Square
Cambridge, Massachusetts 02139

Mr. Richard B. Blue, Sr.
TRW Systems Group
Scientific Data Processing Lab
One Space Park—R3/1098
Redondo Beach, California 90278

Mr. Willard J. Holden
P.O. Box 2278
Sunnyvale, California 94087

IEEE DIRECTORS

Dr. A. S. Hoagland
IBM Corporation
Dept. 29A—Bldg. 910
P.O. Box 1900
Boulder, Colorado 80302

Mr. Thomas N. Pyke, Jr.
Institute for Computer Sciences
and Technology
National Bureau of Standards, Bldg. 225
Washington, D.C. 20234

Dr. S. S. Yau
Department of Computer Science
Northwestern University
Evanston, Illinois 60201

DPMA Directors

Mr. James B. Sutton
Cummins-Allison Corp.
1900-27th Avenue, S.
Homewood, Alabama 35209

Dr. Carl Hammer
Univac Federal Systems
2121 Wisconsin Avenue, N.W.
Washington, D.C. 20007

Mr. Walter A. Johnson
Consolidated Paper Company, Inc.
P.O. Box 50
Wisconsin Rapids, Wisconsin 54494

Society for Computer Simulation Director

Mr. Frank C. Rieman
Computer Sciences Corporation
1101 San Antonio Road—Suite 202
Mountain View, California 94040

American Institute of Aeronautics and Astronautics Director

Dr. Robert R. McCready
Vought Systems Division
P.O. Box 5907
Dallas, Texas 75222

American Statistical Association Director

Mr. James Filliben
Statistical Engineering Laboratory
National Bureau of Standards
Washington, D.C. 20234

Instrument Society of America Director

Mr. Theodore J. Williams
Purdue University
102 Michael Golden
West Lafayette, Indiana 47907

Society for Information Display Director

Dr. C. P. Crocetti
Rome Air Development Center-XP
Griffis Air Force Base, New York 13441

Association for Education Data Systems Director

Dr. Sylvia Charp
The School District of Philadelphia
Board of Education
5th and Luzerne Streets
Philadelphia, Pennsylvania

Association for Computation Linguistics Director

Dr. A. Hood Roberts
Center for Applied Linguistics
1611 N. Kent Street
Arlington, Virginia 22209

American Institute of Certified Public Accountants Director

Mr. Donald Adams
AICPA
666 Fifth Avenue
New York, New York 10019

American Society for Information Science Director

Mr. Robert J. Kyle
1066 McConnell Drive
Decatur, Georgia 30033

Society for Industrial and Applied Mathematics Director

Dr. D. L. Thomsen, Jr.
SIAM Institute for Mathematics and Society
97 Parrish Road, South
New Caanan, Connecticut 06840

Special Libraries Association Director

Mr. Herbert S. White
Stechert-MacMillan, Inc.
7250 Westfield
Pennsauken, New Jersey 08110

Institute of Internal Auditors Director

Mr. William E. Perry
The Institute of Internal Auditors
5500 Diplomat Circle
Orlando, Florida 32810

NATIONAL COMPUTER CONFERENCE BOARD

President

Mr. George Glaser
225 Warren Road
San Mateo, California 94402

Vice President

Mr. Paul W. Berthiaume
Electronic Associates, Inc.
185 Monmouth Park Highway
West Long Branch, New Jersey 07764

Treasurer

Mr. Marvin Ehlers
Square D Company
Executive Plaza
Park Ridge, Illinois 60068

DPMA Representative

Mr. James Case
Dylakor Computer Systems, Inc.
2222 Corinth Avenue
Los Angeles, California 90064

ACM Representative

Dr. Smith Dorsey
Rockwell International
Mail Code AA 54
3370 Miraloma
Anaheim, California 92803

IEEE Representative

Dr. Merlin Smith
T. J. Watson Research Center
P.O. Box 218
Yorktown Heights, New York 10598

SCS Representative

Mr. Ralph Wheeler
Lockheed Missiles & Space Co.
P.O. Box 504
Sunnyvale, California 94088

AFIPS Representative

Dr. Sylvia Charp
The School District of Philadelphia
Board of Education
5th and Luzerne Streets
Philadelphia, Pennsylvania

NATIONAL COMPUTER CONFERENCE COMMITTEE

Mr. Jeffery D. Stein, Chairman
On-Line Business Systems, Inc.
One Embarcadero Center
San Francisco, California 94111

Mr. Jerry L. Koory
9537 Texhoma Avenue
Northridge, California 91324

Dr. M. M. Astrahan
IBM Research Laboratory
Monterey & Cottle Roads
San Jose, California 95193

Dr. Henry S. MacDonald
Bell Labs
Murray Hill, N.J. 07971

Mr. Al Hawkes
Computer Horizons
53 West Jackson Blvd.
Chicago, Illinois 60604

Dr. Harvey L. Garner
Moore School of Electrical Engineering
University of Pennsylvania
Philadelphia, Pennsylvania 19104

Mr. Russell K. Brown
Moore Paper Company
P.O. Box 805
Houston, Texas 77001

1975 NATIONAL COMPUTER CONFERENCE CHAIRMAN

Mr. Donal A. Meier
2756 Mountain View Drive
Escondido, California 92027

1976 NATIONAL COMPUTER CONFERENCE CHAIRMAN

Dr. Carl Hammer
Univac Federal Systems
2121 Wisconsin Avenue, N.W.
Washington, D.C. 20007

1975 NATIONAL COMPUTER CONFERENCE COMMITTEES

General Chairman

Donal A. Meier
Consultant
Escondido, CA

Controller

Lynn Maxson
IBM Corporation
Los Angeles, CA

NCCC Representative

Jerry L. Koory
On-Line Business Systems, Inc.
Beverly Hills, CA

Pioneer Day Coordinator

Ev Bonney
Dartmouth College
Hanover, NH

Exhibits Chairman

Stephen Bowers
General Automation
Anaheim, CA

Local Promotion and Publicity Chairman

Andrea Graham
Xerox Corporation
Inglewood, CA

International Chairman

Patricia MacKenzie
TRW Systems Group
Redondo Beach, CA

Special Events Chairman

Ross Penne
University Computing Center
Los Angeles, CA

Local Arrangements Chairman

Mary Rich
Informatics Mark IV Systems Company
Canoga Park, CA

Registration Chairman

Dennis Sossi
TRW Systems Group
Redondo Beach, CA

TECHNICAL PROGRAM COMMITTEE

Chairman

Stephen W. Miller
Stanford Research Institute
Menlo Park, CA

Vice Chairman

Irwin Derman
National BankAmericard, Inc.
San Mateo, CA

R. Stockton Gaines
The RAND Corporation
Santa Monica, CA

Paul Glaser
Transaction Technology, Inc.
Los Angeles, CA

Paul Malik
United Airlines
San Francisco, CA

Richard A. Marciano
Stanford Research Institute
Menlo Park, CA

Thomas Murray
Del Monte Corporation
San Francisco, CA

Norman Pobanz
Bechtel Corporation
San Francisco, CA

Anthony I. Wasserman
University of California
San Francisco, CA

SPECIAL ASSISTANTS TO THE PROGRAM COMMITTEE

Paul L. Armer
Center for Advanced Study in the Behavioral Sciences,
Stanford, CA

Richard G. Canning
Canning Publications
Vista, CA

David Jasper
Control Data Corporation
Minneapolis, MN

Tosiyasu L. Kunii
University of Tokyo
Tokyo, Japan

Peter Lykos
Illinois Institute of Technology
Chicago, IL

Joseph G. Rubenson
Stanford Research Institute
Washington, D.C.

TECHNICAL PROGRAM AREA DIRECTORS

Vaughn Alexander
Texas Medical Foundation
Austin, TX

Glen Bacon
IBM Corporation
San Jose, CA

Vinton Cerf
Stanford University
Stanford, CA

Edgar F. Codd
IBM Corporation
San Jose, CA

Robert F. Daly
Stanford Research Institute
Menlo Park, CA

John J. Donovan
MIT Sloan School
Cambridge, MA

Ugo O. Gagliardi
Honeywell Information Systems
Waltham, MA

Glyn H. Jones
Burroughs Corporation
Mission Viejo, CA

Earl C. Joseph
Sperry Univac Defense Systems
St. Paul, MN

Gopal Kapur
Consultant
Pleasanton, CA

Theodore Laliotis
Fairchild Systems Technology
San Jose, CA

Donald C. Lincicome
Control Data Corporation
Sunnyvale, CA

Robert Merrell
Burroughs Corporation
Detroit, MI

Richard G. Mills
First National City Bank
New York, NY

Susan Nycum
MacLeod, Fuller, Muir and Godwin
Los Altos, CA

Edward J. Palmer
President, DPMA
Boston University
Boston, MA

Donn B. Parker
Stanford Research Institute
Menlo Park, CA

Bertram Raphael
Stanford Research Institute
Menlo Park, CA

William Stritzler
AT&T
Morristown, NJ

Bruce Wrigley
Travelers Insurance Co.
Hartford, CT

SESSION CHAIRMEN

Alshuk, Thomas J.
Archibald, Julius A., Jr.

Barnett, G. Octo
Barsamian, Harut
Barton, Robert
Bateman, Barry L.
Bell, Thomas E.
Berthiaume, Paul
Bigelow, Robert P.
Blois, Marsden S., Jr.
Brandejs, Jan F.
Buzen, Jeff

Carlisle, James
Case, James A.
Case, Richard P.
Cashman, Thomas J.
Cerf, Vinton
Charp, Sylvia
Codd, Edgar F.
Cohen, Leo J.
Cotton, Ira W.
Crowe, Susan
Curran, Alex
Curtis, Kent K.

Dahm, David M.
Davis, John C.
de Picciotto, Sami
Dolotta, Ted
Donovan, John J.
Dorf, Larry

Eastman, Charles M.
Estrin, Gerald

Farber, David J.
Fife, Dennis W.
Frank, Howard
Freed, Roy N.

Gaines, R. Stockton
Gilchrist, Bruce
Gorry, G. Anthony
Greenfeld, Norton
Guiteras, Joe J.

Hammer, Michael M.

Joseph, Earl C.
Joyce, James

Kahn, Robert E.
Kaspar, Hans
Kay, Alan
Kernighan, Brian
Kolence, Kenneth W.

Laliotis, Ted
Lewis, F. Sherwood
Lindstrom, E. E.
London, Ralph L.
Lowenthal, Eugene I.

Madnick, Stuart E.
McKee, Watson, Jr.
Mills, Richard G.
Moehrke, Don

Morrison, John R.
Moshman, Jack

Nelson, Eldred
Nycum, Susan H.

Oliver, Paul

Patterson, G. Stuart
Paul, Richard L.
Perry, William E.
Plagman, Bernard K.

Rockart, John F.
Russo, Paul M.

Saltzer, Jerome H.
Shelly, Gary B.
Shneiderman, Ben
Smoot, Oliver R.

Taulbee, Orrin E.
Thayer, Richard H.
Thomsen, D. L., Jr.
Traweek, B. Ray
Treu, Siegfried

Uhrbach, Harold

Vallbona, Carlos
Van Trees, Harry L.

Walden, David C.
Walker, Rob
Wasserman, Anthony I.
Weiner, Hesh
Woo, Tony C.
Wortman, David

DISCUSSANTS, MODERATORS AND PANELISTS

Allen, John R.
Altshuler, Gene
Amarel, Saul
Anderson, Robert B.
Anderson, Robert H.
Astrahan, Morton M.

Balzer, Robert W.
Bernacchi, Richard L.
Bernstein, M. I.
Binford, Thomas O.
Bloom, Naomi L.
Bonner, Roy F.
Bower, Cal
Boyd, D. L.
Brandejs, Jan F.
Branscomb, Lewis M.
Brooks, Ruven
Brown, John S.
Brown, Robert R.
Brunner, Theodore F.
Buchanan, Bruce
Burrows, James

Callan, Robert W.
Camuso, John
Candlin, James E.
Carr, Eugene
Chapman, Mary A.
Charney, Jule G.
Clapp, Fritz H.
Connole, Anthony W.
Couperus Jitze
Courtright, Benjamin F.

Davis, Keagle
Denning, Peter
DeRose, James F.
Dickson, R. S., Jr.
Dilligan, Robert J.
Dunn, Robert M.
Dwyer, Tom

Elspas, Bernard
Epstein, Hank
Evans, W. J.

Feigenbaum, Edward A.
Fenwick, William
Feth, George C.
Fields, Craig I.
Fikes, Richard E.
Fletcher, Dennis
Forshay, James
France, N. A.

Garabedian, Robert J.
Gilchrist, Bruce
Gluckson, Fred A.
Goldberg, Adele
Goldman, Jay
Golomb, S. W.
Good, Donald I.
Goodkin, Norman M.
Goodman, David
Gosden, John

Habib, Stanley
Hamming, R. W.
Harder, Donald C.
Hays, William R.
Hopper, Grace M.
Howard, Philip C.
Howe, W. Gerry

Jerman, Max
Johnson, Clay T.
Johnson, Thomas H.
Jordan, Dale
Judenberg, Joseph

Kaplan, Alan
Kapur, Gopal
Karplus, Walter
Keller, Arnold
Kephart, Horace L.
Kirkbride, Chalmer G.
Kirkley, John
Kiviat, Philip J.

Lee, Jerome
Lewis, Don
Lilly, Don
Liskov, Barbara H.
Litherland, H. K.
Lord, Kenniston W., Jr.
Lubert, Marvin
Luebbert, William F.
Lundell, Drake
Lynch, William C.

Madden, William
Maloney, James C.
Martin, William A.
McLeod, Dennis, J.
Merchant, Eugene
Metz, Douglas
Miller, Donald S.
Miller, Terry
Miller, William F.
Moffett, Thurber J.
Muntz, Richard

Nevins, James L.
Notley, M. Garth
Nunley, Leonard J.

Otto, William

Papert, Seymour

Ream, Norman
Rice, Rex
Ritchie, Robert W.
Robbins, Galen P.
Roland, Donald
Rosen, Charles A.
Rosenblatt, B. A.
Rowe, Barry D.
Rubey, Raymond

Scanlon, Robert
Schlesinger, Stewart
Schneidman, Arnold
Shuey, Richard L.
Smagorinsky, Joseph
Smith, D. M.
Sondak, Norman
Standish, Thomas A.
Steel, T. B., Jr.
Stonebraker, Michael
Suchoff, Benjamin

Testa, Charles
Tsichritzis, Dionysios

Uncapher, Keith

Vick, Charles
Viemeister, Peter E.

Walls, Dale
Ware, Willis H.
Wegbreit, Ben
West, David
Wessler, Barry D.
Whidden, Phillips
White, George R.
Wilner, Wayne T.
Wilson, Kent
Wilson, Robert G.
Witte, Juergen
Worsley, Alice F.

Ying, Charles
Yourdon, Edward
Yovits, Marshall

Zahn, Charles T.
Zilles, Stephen
Zobrist, Dale W.

REVIEWERS

Abrams, Marshall D.
Alexander, Vaughn
Allan, John J.
Alshuk, Thomas J.
Archibald, Julius A., Jr.
Aron, J. D.
Arterbery, Vivian
Atwood, Delbert
Aupperle, Eric M.

Bacon, Glen
Baer, J. L.
Baker, Robert L.
Barnett, G. Octo
Barr, Avron
Barsamian, Harut
Bateman, Barry L.
Beaudrex, Paul
Beckett, J. Terry
Bedard, F.
Belady, L. A.
Bell, Thomas E.
Bemer, Robert
Beradino, Al
Berra, P. Bruce
Blackmore, Steven J.
Blanc, Robert P.
Blois, Marsden Scott, Jr.
Boehm, Barry W.
Bonine, Ken C.
Booth, Taylor L.
Bouknight, W. Jack
Bowie, J. R.
Branco, Cosmo
Brandejs, Jan F.
Brandon, Dan
Branstad, Dennis K.
Brooks, Ruven
Brown, Russell
Burchfiel, Jerry D.
Burdge, Geoffrey L.
Burke, Ed.
Butler, James M.
Buzen, Jeffrey P.

Campaigne, Howard H.
Canning, Richard G.
Cardenas, Alfonso F.
Carlisle, James
Carlson, Eric D.
Carter, William C.
Case, Leon R., II
Casey, Richard G.
Cashman, Thomas J.
Cerf, Vinton
Chamberlin, Donald D.
Charp, Sylvia
Cheek, Robert C.
Chen, Tien Chi
Chu, Wesley W.
Chu, Yaohan
Clapp, Fritz H.
Codd, Edgar F.
Corley, Melvin R.
Cotton, Ira W.
Couperus, Jitze
Courtright, Benjamin F.
Crowe, Susan
Curran, Alex

Dahm, David M.
Daly, Robert F.
Davidson, Charles H.
Davidson, Ed
Davis, John C.
Donovan, John J.
Dorf, Larry
Douglas, John
Drane, Douglas
Dudgeon, D.
Duncan, Karen A.

Eastman, Charles M.
Eckhouse, Richard H., Jr.
Ehardt, Joseph L.
Elman, Stanley A.
Elspas, Bernard
Estrin, Gerald

Fabry, R. S.
Falk, G.
Farber, David J.
Feng, Tse-yun
Ferrari, Domenico
Fife, Dennis W.
Fikes, Richard E.
Foster, Caxton C.
Frank, Howard

Gagliardi, Ugo O.
Gaines, R. Stockton
Gantner, George E.,
Gault, Charles
Gibb, Kenneth R.
Giloth, Paul K.
Gimball, James
Glaseman, Steven
Glaser, Paul
Glick, Norman
Glorioso, Robert M.
Goldberg, Adele
Goldberg, Mark
Golomb, Solomon W.
Gorman, Don
Grampp, F. T.
Gray, James N.
Greenblott, Bernard J.
Greenes, Robert A.
Greenfeld, Norton R.
Guiteras, Joseph J.

Habermann, Nico
Habib, Stanley
Hammer, Carl
Hammer, Michael M.
Hamming, Richard W.
Harmon, John C.
Harrison, Malcolm C.
Harter, M. D.
Hawyrlko, Warren P.
Helgeson, Duane
Hernon, James A.
Highland, H. J.
Hoffman, Lance
Hollingworth, Dennis
Hong, Se June
Hook, Harvey O.
Hopper, Grace M.
Horne, William J.
Houston, Rick
Hsiao, David K.
Hsiao, Mu Y.
Hudak, John

Jacks, Edwin
Jones, Glyn H.
Joseph, Earl C.
Joyce, James

Kahn, Robert E.
Kandel, Abraham
Kapur, Gopal K.
Karplus, Walter
Kaspar, Hans
Kay, Alan C.
Keller, Robert M.
Kelly, John
Kernighan, Brian W.
Kimbleton, Stephen R.
Kimme, Charles
King, W. Frank
Kirshenbaum, Frank
Kleir, Richard
Koeppen, Karl
Kolence, Kenneth W.
Kozik, Eugene
Krause, Kurth W.
Kroeger, Joseph
Kroeger, Joseph
Kunii, Tosiyasu L.
Kuo, Frank F.

Laliotis, Theodore

Lawrie, D. H.
Lazar, Leonard M.
Ledin, Victor
Lee, Jerome
Lee, John A. N.
Lincicome, Donald C.
Linden, Theodore A.
Lindstrom, E. E.
Link, C. H.
Lipow, Myron
Lippincott, Phil
Lippman, Michael D.
Liskov, Barbara H.
Liu, Jane
Liu, Leonard Y.
Lividini, Joseph
Lorie, Raymond A.
Lowenthal, Eugene I.
Lucus, Brian G.
Ludwig, Herbert
Lum, Vincent Y.

Madnick, Stuart E.
Madron, Beverly B.
Maloney, James C.
Manning, Eric
Manola, Frank
Marcantonio, Angelo R.
Marrigan, Robert J.
McCluskey, Edward J.
McDonald, Clement
McElvain, Kay
McFarland, Clay
McKee, Watson M., Jr.
McKeeman, W. M.
McLeod, Dennis J.
Mehl, James W.
Merrell, Robert
Merwin, Richard E.
Metcalfe, Robert
Miller, Charles
Miller, Stephen W.
Mills, Richard G.
Minsky, Naftaly
Moehrke, Don
Montgomery, Christine
Morrison, John R.
Moshman, Jack
Murray, Edward
Myers, Willard L.

Nelson, Eldred
Nelson, John M.
Nicols, A. J.
Nilsen, Ragnar
Nycum, Susan H.

Oliver, Paul
O'Neil, Nathan D.
Osher, William
Osterwell, L. J.
O'Toole, James A.

Palermo, Frank P.
Palmer, Edward J.
Parish, Randall Mel
Parke, Benjamin G.
Parker, Donn B.
Pehrson, Dave
Perry, William E.
Pickens, Kenneth E.
Pinkston, John
Pitts, Gerald N.
Plauger, P. J.
Popek, Gerald J.
Postel, Jon
Prescott, Lee R.
Press, Barry
Prokop, J. S.
Purdy, J. Gerry

Ramamoorthy, C. V.
Raphael, Bertram
Rauscher, Tomlinson
Reines, Jose
Reisner, Phyllis
Reiss, Russell A.
Rettberg, R.
Riedel, Earl W.
Riggs, George
Ringuette, Robert J.
Ritchie, D. M.
Rittersbach, George H.
Rockart, John F.
Rodriquez-Rosell, Juan
Roland, Donald E.
Russo, Paul M.

Saltzer, Jerome H.
Sassenfeld, Helmut M.
Scanlon, J. M.
Schlegel, C. T.
Schoen, Alfred
Schroeder, Michael D.
Schroeder, Russell W.
Schultz, Gaymont
Schuster, Stewart A.
Scott, J. L.
Scott-Morton, Michael
Secrest, Richard D.
Sevcik, K. C.
Shelly, Gary B.
Shiao, D. K.
Shneiderman, Ben
Short, Gerald E.
Sibley, E. H.
Sickel, Sharon
Siewiorek, Dan
Simmons, Richard B.
Sloan, Martha E.
Smith, Diane
Smith, Eugene B.
Snyder, Art
Squires, Stephen L.
Stanford, W. Donald
Stanton, Michael
Stonebraker, Michael
Stritzler, William
Swenson, Richard J.
Szygenda, S. A.
Szymanski, Thomas G.

Taulbee, Orrin E.
Taylor, Robert
Thayer, Richard H.
Thompson, Howard K.
Thomsen, Donald L., Jr.
Toong, Hoo-Min D.
Traiger, Irving L.
Treu, Siegfried
Tsichritzis, Dionysius
Tuel, William G.
Tung, F.
Turn, Rein
Turpin, Terry

Vallbona, Carlos
Van Trees, Harry L.
Varaya, Pravin
Vick, Charles

Waite, William M.
Wakerly, John
Walden, David C.
Walford, Robert B.
Walker, Rob
Wasserman, Anthony I.
Watson, W. Joe
Watt, Jim
Weingarten, Frederick W.
Welker, Nancy
Wexelblat, R. L.
Wicheal, Don
Williams, Richard
Wilner, Wayne T.
Winkler, Stanley
Wiorkowski, Gabrielle
Withington, Frederic G.
Wolf, J.
Wong, E. E.
Woo, Tony C.
Woods, W. A.
Wortman, David B.
Wrigley, Bruce
Wulf, William A.

AUTHOR INDEX