Model 8000
Multiprocessor
System

Theory of Operations and Technical Summary

BTI COMPUTER SYSTEMS

BTI 8000

THEORY OF OPERATIONS AND TECHNICAL SUMMARY

September 1978

PRELIMINARY DOCUMENT

BTI COMPUTER SYSTEMS

CONTENTS

PREFACE

This manual presents general information about the BTI 8000 computer system. The introductory material may be read as a management summary; the manual as a whole should serve the technical reader as a complete source of information for system evaluation and familiarization.

Although some detailed specifications are included, the primary orientation of this manual is to present a conceptual overview, with emphasis on design principles of the system.

THIS IS A PRELIMINARY DOCUMENT, FOR LIMITED DISTRIBUTION ONLY.

The final version of this document should be released later this year. The manual will be rewritten, possibly reorganized, and typeset to improve its readability, with graphic material included. Further information may also be added.

September 1978

Sam Cohen
Shirley Henry

1. INTRODUCTION

1.1 Overview

(PHOTO)

The BTI 8000 is a 32-bit high-performance multi-processor, multi-user, multi-language, and multi-function computing system. Its unique architecture can be scaled to serve in a range of applications between those of a large minicomputer and a mid-to-large size mainframe computer.

The BTI 8000 is the result of over four years of research and development in the application of rapidly evolving hardware and software technology to business information processing.

Advantages of the BTI 8000 computer system include:

MULTI-PROCESSOR:	*UP TO EIGHT 32-BIT PROCESSORS OPERATING IN PARALLEL*
MULTI-USER:	*UP TO 512 SIMULTANEOUS INTERACTIVE TERMINAL USERS (BLOCK MODE, TO 19,200 BAUD), PLUS BATCH*
MULTI-LANGUAGE:	*PASCAL-X COBOL FORTRAN RPG II BASIC-X DBMS-X*
MULTI-FUNCTION:	*INTERACTIVE TURNKEY INTERACTIVE DEVELOPMENT FAIL-SOFT TRANSACTION PROCESSING MULTI-STREAM BATCH SECURE TIMESHARING*

MULTI-FUNCTION PROCESSING

The system is designed for the cost-effective support of a number of distinct simultaneous activities. Larger configurations can concurrently run hundreds of interactive terminals and batch jobs. Activities that are particularly well supported include:

- Interactive turn-key applications, including data entry, data retrieval, and fail-soft transaction processing
- Multi-user database applications
- Interactive program development of both batch and on-line programs, for all languages
- Multi-stream spooled batch processing
- High security on-line applications
- General timesharing, with full accounting and management control
- Multiple-language programming for commercial applications (COBOL, RPG II, FORTRAN); scientific/engineering applications (FORTRAN, PASCAL-X); structured programming (PASCAL-X, COBOL); and educational use (BASIC-X, PASCAL-X)

APPLICATIONS ORIENTATION

The BTI 8000 is the first computer system whose software was designed before its hardware, so that user applications and management needs are serviced efficiently and intelligently at moderate cost.

The machine instruction set was specifically designed for maximum efficiency in the execution of compiler-generated code, as opposed to hand-generated assembly code. By sharing both code and data areas in memory among multiple users, the operating system minimizes disk access, and hence maximizes throughput.

Programmers are isolated from machine details and the complexities of system internals, and are free to concentrate on the design and development of their applications.

- Programs written in all languages address terminals as ordinary I/O devices. There is no need to deal with special communications software.
- A universal interactive editor minimizes program preparation and modification time.
- A single, easy-to-use command and control language allows convenient testing of any program, including batch jobs, from interactive terminals. All development may be interactive.
- Interactive symbolic debuggers totally eliminate the need for tedious core-dump analysis. Symbolic information is permanently linked with object code to allow first-stage debugging of any program without recompilation.
- Device-independent I/O eliminates reprogramming when the application requires different files or even different peripheral devices.
- Dynamically expandable files and automatic disk organization eliminate the need to manage physical "extents".
- Programming languages are extensions of the most comprehensive and recent industry standards. Programmers adapt easily to the system, and outside software packages are easily converted. Languages provided are: COBOL, FORTRAN, PASCAL-X, RPG II, and BASIC-X.
- The PASCAL-X application language (extended PASCAL) is designed specifically for productive structured programming, and generates extremely efficient object code.
- A CODASYL data base management system, DBMS-X, provides uniquely error-protected transaction processing through COBOL, FORTRAN, PASCAL-X, and its own interactive data base language. DBMS-X relieves the data base administrator of all physical data management chores.

UNIQUELY MODULAR ARCHITECTURE

The unique system architecture minimizes risk of purchase and totally eliminates upgrade conversions.

The BTI 8000's Variable Resource Architecture is based upon significant BTI engineering patents. The computer is composed of four types of resource modules, each of which serves an independent architectural function: CPU, memory, I/O, and coordination. The key to this design is a central bus that operates at 60 megabytes per second.

Variable Resource Architecture, in conjunction with its integrated operating system, provides several unique economic benefits:

- *By varying the total number of resource modules (up to 16), a configuration may be selected to match the scale of the application exactly. Neither too much nor too little computer power need be purchased. Configurations can be chosen from under $100,000 to over $1,000,000.*

- *By varying the mix of the different resource modules, a configuration may be selected to match the nature of the application exactly. The customer need not pay for excess system capacity merely to support a special architectural requirement (such as a large number of terminals).*

- *A given configuration is instantly expandable with no change in software. Additional resource modules, including CPU's, are merely plugged into the bus. After plug-in, sysgen/restart is an automatic one-pushbutton procedure that the system accomplishes in seconds.*

  The operating system software is tightly integrated with the architectural design to provide a consistent environment for all user software, across all configurations. This "virtual machine" technology means that no reprogramming, recompilation, or DP staff effort of any kind is required to accommodate a change in configuration. (In fact, programs are deliberately kept unaware of hardware configuration.)

  Application software investment is fully protected as the workload requirements grow: the largest BTI 8000 can use the same programs developed on the smallest. Capacity upgrades are painless and risk-free.

MANAGEMENT CONTROL AND SYSTEM SECURITY

Control and security structures are organic elements of the BTI 8000.

The system is fundamentally and positively secure. Interactive users cannot damage the operating system, or interfere with either system operations or each other. Data files and programs are totally private unless explicitly declared otherwise. Production work may safely be run concurrently with general timesharing.

- System management is automated, so that the system manager need not be a systems programmer. His responsibilities are in applications control rather than technical maintenance.

- System operation is segregated from system management, so that operators cannot access user data without explicit authorization.

- Management can be delegated through a four-level hierarchy of control and data access privilege. Unique positions in this structure can be created for all individuals using the system, with password-controlled access.

- Individual users, or groups of users, can have their interactive environments tailored to separate turn-key applications, whether rote data entry or on-line management information inquiry.

- Applications are controlled and coordinated by the organization of all programs and data files into various separate libraries; these are normally private to an individual user, but may be made public to different groups of users or to the whole system. User privacy is the default state, with specific action required to make data accessible to other users, so that security is not breached through oversight.

- Management authority includes the ability to authorize user access, grant specific allocations of system resources (time and file space) to subordinates within the hierarchy, get full reports on user activities, and limit the ability of subordinates to share data outside their groups.

- The BTI 8000 also specifically deals with the "Trojan Horse", "false front", and other security problems in multi-user on-line computing.

HIGH SYSTEM AVAILABILITY

In fault detection, tolerance, and recovery, the first priority of the system is to preserve data integrity; the second is to preserve operational continuity.

- Variable Resource Architecture is fail-soft. If one of several CPU's or memory modules fails, and no replacement is available on-site, the system can be restarted in its reduced configuration. The same job mix is still serviced at the reduced performance level.

- The hardware detects errors through integrated multiple self-checking mechanisms. Inconsistent operation immediately removes a resource module from service, with the operator notified.

- Further diagnosis is immediately available through on-line remote communications with the BTI service computer at the BTI national service center. This facility is also used for on-line factory software maintenance. The link can be disabled at the front panel, if desired.

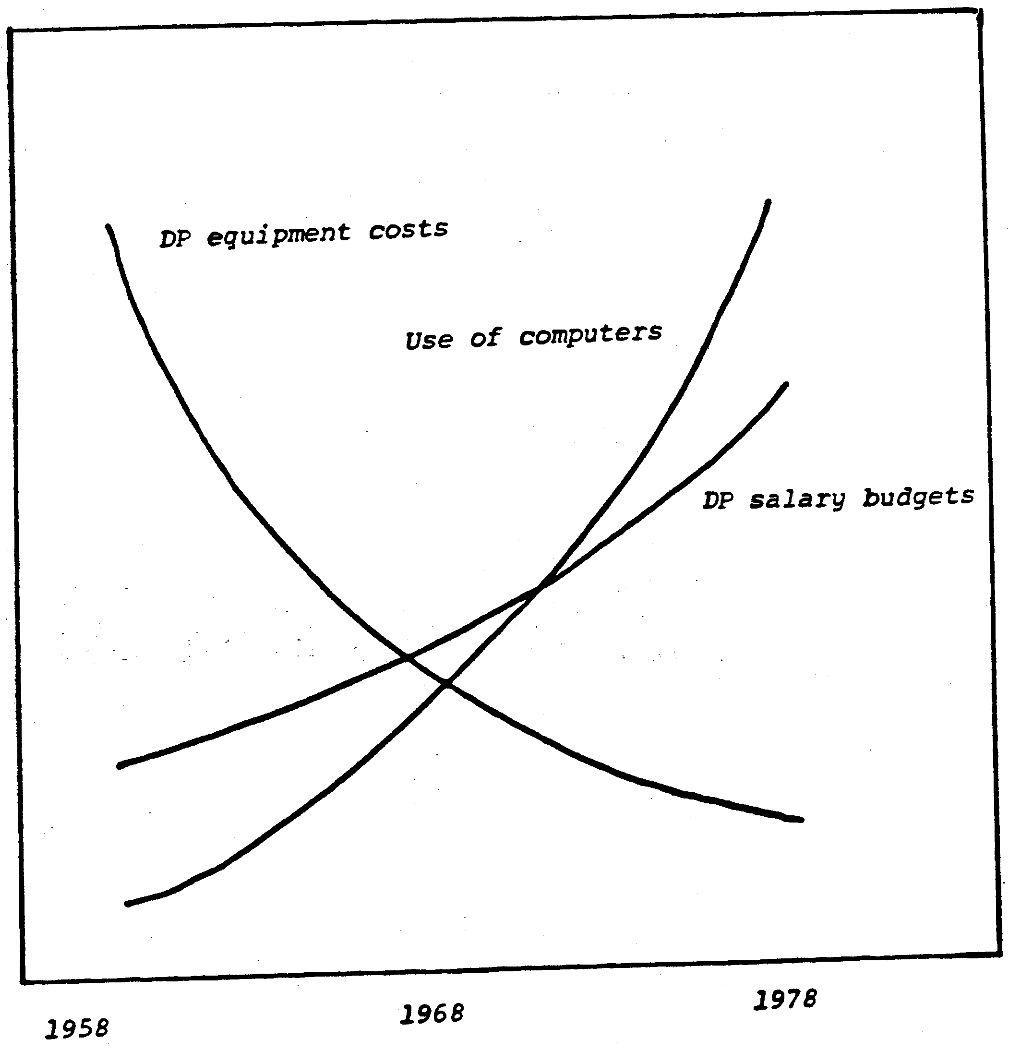
- Recovery from power outages is automatic and does not require operator assistance; all operations resume where they were interrupted. A self-contained battery pack ensures that system date and time are correct, even after a full week.

- If the communication link with a remote terminal is lost during an interactive session, the system closes out the session in an orderly manner. Explicit line disconnect routines can also be programmed to back out partial transactions, write checkpoints to files, etc.

- A disk data reconstruction facility automatically rebuilds most data blocks whose contents are partially destroyed by bit or burst errors, including physical disk problems. This mechanism is unique to the BTI 8000, and is patented by BTI. The system is thus protected against the most common cause of data loss and system crashes.

- The data base management system uses unique software technology to preserve the consistency of data bases at all times, even in the event of a system crash during an update transaction. DBMS-X also provides for incremental data base backup, audit-trail and journal logging, and the creation of test data bases to allow for fail-safe program development.

1.2 System Philosophy



The design philosophies of the BTI 8000 are based on several observations about business information processing economics:

1. *Data is an ever more valuable organizational resource.*
2. *Data processing is an ever more critical organizational function.*
3. *DP equipment costs are decreasing while people costs increase.*

The BTI 8000 is an on-line system, designed for large-scale interactive multi-user capability. Larger configurations can simultaneously run hundreds of terminal users, plus multi-stream batch jobs.

Traditional data entry and data retrieval procedures (keypunch and lengthy print-outs) are substantially less cost-effective for an organization than on-line, interactive terminal access. While some detailed paper reports are always necessary, the indirect costs of not having timely information immediately and easily available to the end-user are even more significant than the direct costs of keypunch and support operations.

The BTI 8000 protects data against damage through fail-soft data handling and storage. Mechanisms include:

- Internal validity checking
- I/O transfer validity checking
- Disk error correction/ data reconstruction
- Crash-proof DBMS update transactions

The BTI 8000 provides for full data security, privacy, and protection in a multi-user environment, through:

- Non-interference among user activities
- Protection of each user's private data from other users
- Safeguard controls in sharing of data access among users
- Password-protected system access, with audit trails

Data loss, damage, or theft may have financial consequences at least as severe as those caused by the loss, damage, or theft of any other organizational asset. Even if certain data is not considered critical, replacing it if destroyed requires the application of people, time, and other resources. Security problems with sensitive data can seriously damage an organization as a whole.

The BTI 8000 is designed to be a fail-soft, redundant system through the use of multiple, pooled hardware resources.

The system uses a multi-level approach to hardware diagnosis and correction, through automatic self-diagnosis plus a remote, on-line link to factory service.

System operations are protected: the operating system is insulated from user activities.

Unavailability of the data processing resource can immobilize many organizations. Most computer systems are totally inoperable if one of their major components fails; then, fault diagnosis and correction are dependent on the availability of vendor service personnel. Computers are also traditionally vulnerable to certain types of programming errors, where the operating system is accessible to users.

The BTI 8000 provides an automated environment for application development and execution. The system allows DP management to focus attention on applications, by relieving them of the burden of managing system internals.

The expenses of recruiting and compensating high-quality, talented data processing technical personnel comprise an enormous portion of the DP budget. A surprisingly large percentage of this resource, however, is traditionally employed in the servicing, enhancement, and operation of the computer system itself, rather than in the application of the computer to the purposes of the organization. It is not uncommon to have thirty percent of a programming staff engaged in the so-called systems programming function. It is a costly irony of technology that the very source of automation is itself so little automated.

The BTI 8000 is a cost-effective vehicle for applications programming; its facilities include:

- End-user interactive turn-key programming tools
- Full interactive program development and test facilities
- Standard forms of major programming languages: COBOL, FORTRAN, PASCAL-X, RPG II, and BASIC-X
- A full-scale CODASYL-type data base management system

Application software costs are synonymous with the people costs of programming. Minimizing the need for programming, and the effort required to achieve stated programming goals, by definition minimizes this ever more expensive cost. There are several areas of concern in developing (or acquiring), validating, and maintaining application software:

(a) Distance of users from data. If users do not have direct access through on-line terminals for data inquiry, programmer time is constantly required to translate their requests to the computer.

(b) Efficiency of programming procedures. On-line interactive program development and testing, time-shared with production operation of the computer, is enormously more effective than the traditional batch-processing approach.

(c) Standardization of programming languages. Use of industry-standard versions of the major programming languages minimizes the costs of programmer recruiting and training, and reduces or eliminates the cost of converting purchased software packages. Use of non-standard languages may preclude the possibility of make/buy decisions.

(d) Use of data base technology. Multiple, uncoordinated uses of the organizational data resource lead to large program maintenance costs as the complexity of applications increases and as new forms of data are added. Data base technology provides a single, centrally controlled organization for data, and a maintenance-free interface to it for existing and future application programs.

The BTI 8000 is characterized by its unique hardware and software technology: Variable Resource Architecture and Virtual Machine Multiprocessing. Together, they provide incrementally expandable system capacity to meet growth in workload without DP staff effort. The VRA/VMM philosophy includes:

- Plug-in hardware resource expansion
- Pushbutton automatic SYSGEN
- Hardware configuration totally transparent to all software: no reprogramming, recompilation, or tuning is necessary when the system configuration is changed

As the general economy expands, the growth of DP activity inevitably leads to the exhaustion of of computer capacity and the need to upgrade the equipment to accommodate the growing workload. Because of constant advances in technology, resale value of used DP equipment tends to fall off rapidly with time, while the economic risk of the new capital investment decision increases. Furthermore, the upgrade traditionally results in a major cost in time, effort, and parallel operation for converting the existing application software to run on the new system, with secondary costs in the risk of erroneous operation of improperly converted programs.

16

1.3 Integrated System Design

The BTI 8000 is a high-performance 32-bit multiprocessor computing system that couples a virtual machine multiprocessing monitor to a uniquely modular hardware architecture. (The term "virtual machine" refers to the environment in which programs run. The characteristics of this environment need not match those of the physical machine on which it is implemented.)

The central design principle is separation of function. At the conceptual level, application program logic and application programming effort are separated from both the size of the workload -- i.e., the amount of program use -- and the physical system configuration selected to support that workload. At the architectural level, performance/cost ratio of the system is maximized by balancing and distributing architectural functions among a mix of specialized hardware modules. The distribution of functions -- i.e., the relative mix of the different types of modules -- may be selected to match the nature of the workload; the amount of system capacity -- i.e., the total number of modules that comprise the computer -- may be selected to match the size of the workload. The design is therefore known as Variable Resource Architecture.

The design assigns separate but cooperative responsibility to one or more resource modules for each of three basic functions:

- Execution of program instructions;
- Access to main memory;
- Movement of data through input/output channels.

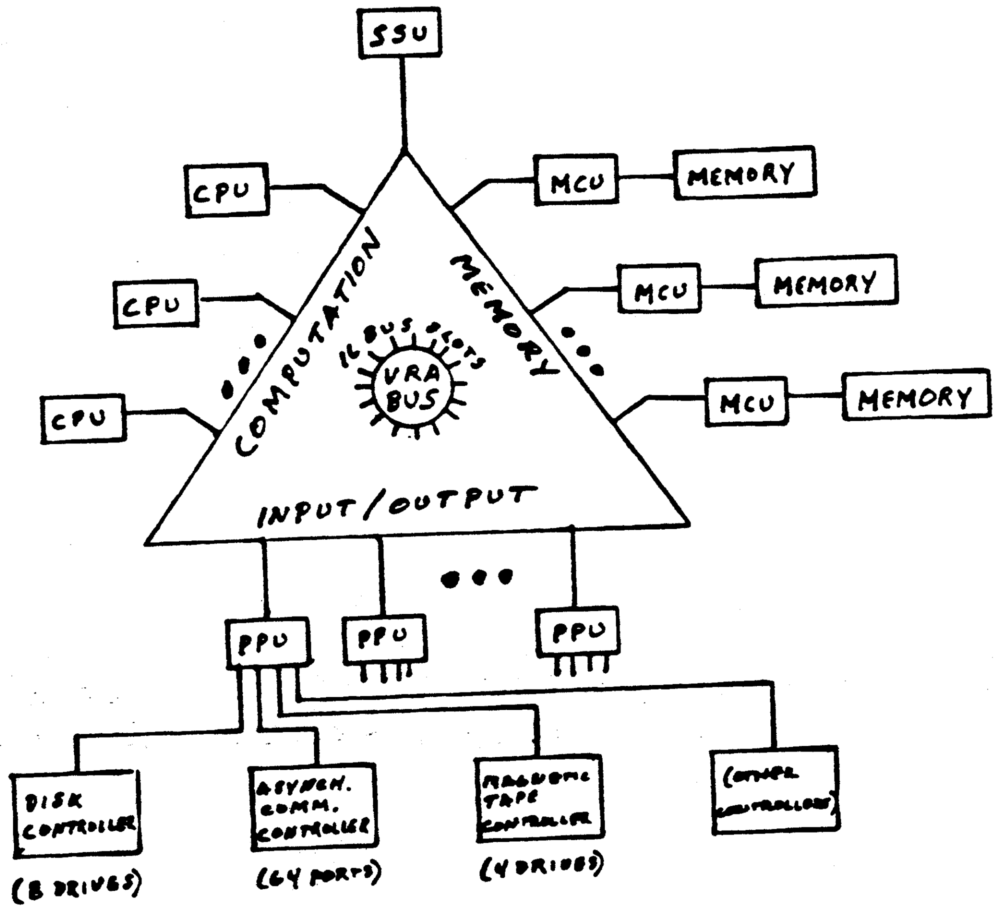
A fourth module type is responsible for internal system services and coordination.

BTI has applied for several major hardware patents on a central interconnection mechanism that makes this unique architecture possible: The internal VRA bus is a 32-bit-wide path through which data passes between resource modules at the rate of 60 megabytes per second. Its ultra-high transport rate, four times faster than possible with previously available technology, enables the separate components to function in parallel as an integrated system, since inter-module communication is seldom delayed by bus contention. For example, a burst data transfer from a storage module disk drive occupies only four percent of available bus capacity.

The integrated virtual machine monitor pools the available hardware resources into a single physical machine, which it then uses to implement and support separate virtual machine environments for each "process", or user activity, on the system.

The result is that all software operations on the system use the same, well-defined virtual machine specification, independent of the hardware mix or size. Application programs are written without concern for configuration, and will execute unchanged on any BTI 8000 system, from smallest to largest. Differences in the size of a system affect only overall work/time ratio, not the user environment or system services available to user processes.

Since the system software was designed before the system hardware, the latter provides direct and specific support of Monitor functions to maximize virtual machine efficiency and hence overall performance/cost ratio.



SEPARATION OF FUNCTION: VARIABLE RESOURCE ARCHITECTURE

CPU:	Computational Processing Unit
MCU:	Memory Control Unit
PPU:	Peripheral Processing Unit
SSU:	System Services Unit

1.4 Process Services

A "process" is the distinct execution of a program. Several processes may arise from the same program, if that program is invoked by multiple users concurrently. The BTI 8000 Monitor allows multiple concurrent users to share the same copy of a common program in memory. The system can support hundreds of processes simultaneously, using any number of shared or individual programs. The environment in which each process runs is its "virtual machine". Virtual machine specifications are independent of physical machine specifications. A "processor" is the hardware module that executes program instructions. Processors run processes.

The system provides the following services to user processes:

Computation:

Processors own neither memory nor processes: assuming that there are at least as many processes to be run as processors to run them, the system assigns each processor to some runnable process until either a roadblock condition arises (such that the processor would be idle until some external event ends the roadblock), or the assigned time quantum expires. At this point the process is switched out of that processor, and the processor is reassigned to the most eligible waiting process.

There are no master-slave relationships among processors, and they operate independently of one another; a processor is a computational resource in the service of processes. Monitor functions, including process scheduling, "float" among processors in the same manner as user processes.

The machine language instruction set is specifically designed to efficiently execute compiler-generated code, as opposed to the hand-generated assembly code which matches the instruction set of traditional machines. There is direct processor support of the data structures used in compiled code: stacks, queues, arrays, and linked lists, with arbitrary-size data elements. Thirty-two address modes provide for efficient manipulation of these structures, whose data elements may be character strings, arithmetic or logical operands, pointers, bit fields, etc. Subroutine linkage, with parameters passed by address or by value, is facilitated by a special portion of the instruction set.

Arithmetic operations use single-word (32 bits) or double-word (64 bits) integer operands, 64-bit floating-point operands, plus (with system software support), 128-bit double-precision floating-point operands. Processor logic also includes facilities for operand exchanges and simultaneous storage of results in both memory and registers.

Memory:

Each process owns one-half megabyte (512 kb) of program address space, independent of the amount or configuration of physical memory, or of the number of processes on the system. Memory is managed in fixed size pages of 1024 32-bit words (thus the virtual memory space of each process is 128 pages). Process pages float between physical main memory and mass storage (disk) transparently to the user processes; the system uses a modified demand-paging technique that precludes thrashing.

BTI 8000 design essentially eliminates the overhead time of virtual-to-physical page address translation, which is the traditional limiting factor in virtual memory design. When the system assigns a processor to a process, the current translation table for that process is loaded into a special fast-access memory built into the processor hardware; subsequent translation requires only 67 nanoseconds, a small fraction of instruction execution time.

The system optimizes the use of physical memory by sharing data pages as well as code pages among processes, whenever possible. If several interactive terminals are using the same program, only one copy of each program page need be in memory; if one of the processes modifies a writable program page (an "impure" page), the Monitor creates a private copy for modification, and the other processes continue sharing the original page.

Files and I/O:

The mass storage resource (disk) supports internal system operation, virtual memory, and the file system, which includes logical devices. All program I/O is performed through virtual I/O channels; each process can use up to 202 virtual channels. At run time, virtual channels are associated with logical devices, which may be various types of mass storage files (sequential access files, random access files, executable code files); physical device analogs (magnetic tape, line printers, the interactive terminal, etc.); or other logical devices, including inter-process communication mechanisms. All physical devices, with the exception of terminals and magnetic tapes, are automatically spooled. Batch processing is facilitated through spooled input (virtual card reader) and output (line printer) queues.

Since users may perform virtual channel assignment externally to programs, even physical peripheral configuration need not affect program independence. For example, a program which generates an output stream of character records may be variously used to output to a line printer, magnetic tape, disk file, or terminal.

At the hardware level, peripheral I/O is controlled by intelligent high-speed channels whose operation and access to memory is independent of program computation. The Monitor intercepts program I/O requests and issues the appropriate privileged instructions to the channels.

<u>Privacy/security</u>:

Each process operates under the auspices of an account, the locus of resource use and data ownership. There may be thousands of accounts on a system. Accounts are allocated system resources (processor time, wall-clock time, mass storage) in a four-level control hierarchy, and are charged for their use. Each account may own a private mass storage library of programs and files.

The system is basically closed and passively secure; that is, a user must have permission to use the system in the first place, and then to access private resources outside his ownership. The activities of all users, including system operators, are confined to their private environments, unless other accounts have granted access permission either explicitly or through a supervisor-subordinate relationship or public library.

Access to programs, files, and inter-process communication facilities can be restricted in several dimensions, and can include an access-time password requirement. A process can even restrict the execution of certain critical program operations prior to running a suspect borrowed program, or during debugging.

The virtual machine process environment allows direct execution of only those processor instructions which deal with the virtual memory address space. All other operations, including I/O, are performed through "executive request" pseudo-instructions which the processors trap out, so the Monitor can service the request safely. Each process is protected from interference from other processes, and the system itself is protected from harm from any process.

1.5 Programmer Services

BTI-supplied software provides program development services based on the following assumptions:

- o The users are not system programmers, but are competent in the use of one of the major programming languages (COBOL, FORTRAN, etc.) and in knowledge of the application.

- o Many of the applications being programmed are commercial systems, characterized by database use, sensitive data, and an interactive terminal interface to a turnkey end-user unfamiliar with computers.

- o The DP organization should be able to take advantage of existing application packages written in a standard form of one of the major programming languages.

Hence the following services are provided for the programmer:

Control Mode:

The interactive command language is familiar, easy to learn, and neither complex nor introverted in design. The user is not involved in internal system operations. The same Control Mode language is also used for job stream control and batch processing. If desired, special-purpose control languages may be programmed for application to one or more accounts.

Data security:

The file system includes carefully designed mechanisms to prevent unauthorized access to data. Unless specifically created for inter-machine transportability, data is recorded in an encrypted form on all portable magnetic media. The system manager can prevent even system operators from accessing sensitive data, such as payroll files. Project and division managers can prevent their subordinates from allowing data to be accessed by users outside the project or division.

Crash resistance:

The system is designed with a commitment not to lose data because of system failure. Data structures are crash-resistant, in the sense that operations can be interrupted at any point and the structures are still validly accessible. At a higher level, the DBMS-X software facility provides crash-proof transaction processing of databases: in case of failure or interruption, the most recent update transaction is posted either completely or not at all, so that the entire database remains coherent at all times.

Interactive development:

Programs are normally developed on interactive terminals, which can operate at rates up to 19,200 baud. Program preparation and modification is done with a BTI-supplied interactive editor, or (for BASIC-X) an incremental compiler. Interactive symbolic debuggers are provided for all languages, and can be used for symbolic variable examination of programs that have not been compiled specifically for debugging.

Turnkey facilities:

All programming languages allow the programmer to assume full control over end-user terminal interfaces. Programs can control formatting, error recovery, and terminal interrupts, so that a data entry clerk, for example (or a financial manager) is not confronted with any messages except those planned by the programmer. The end-user of any account can be placed under this kind of fully controlled turnkey environment from session sign-on to exit.

Full-scale CODASYL-type DBMS:

Application designers may choose to store data under the control of a BTI-supplied database management system (DBMS). DBMS-X extends the CODASYL specifications for multi-level network databases, and is accessible through COBOL, FORTRAN, and PASCAL-X.

DBMS-X provides unique facilities to specifically protect the structural and logical integrity of databases in a multi-language, multi-user, on-line system. Access Control Lists secure information at the database, dataset, and data item (field) levels. DBMS-X also includes a comprehensive set of utility functions to aid in the creation, backup, restructuring, auditing, journalizing, and recovery of databases.

A separate offering, the Interactive Database Language (IDBL), uses DBMS-X to provide general-purpose interactive database query, update, and data entry without the need to write a user program in one of the application languages for these purposes.

Standard programming languages:

The BTI 8000 language compilers and subsystems are designed to offer a comprehensive implementation of accepted industry standards. This approach allows programmers to become productive on the system quickly, and allows existing application software to run on the BTI 8000 with little or no conversion.

All compilers (and the assembler) generate object code in a standard, reentrant format so that the link-loader can combine routines written in various source languages into a coherent executable program. Symbolic information is carried with executable code to provide an interface with source-level interactive debuggers. Finally, all languages interface to a common file system and support terminals as standard I/0 devices, so that interactive programs can be written in all languages with minimum programmer training required.

PASCAL-X is an extended version of standard PASCAL, a language which allows and promotes structuring of both programs and data. It is remarkably efficient both in terms of programmer productivity and machine execution; PASCAL-X programs generate code that executes as well as good assembly code, at a small fraction of the cost in programmer time. BTI 8000 compilers and utilities are themselves implemented in PASCAL-X.

FORTRAN is an extended version of 1977 ANSI-standard FORTRAN, as accepted by the U.S. Navy and Air Force.

COBOL is an extended version of 1974 ANSI-standard COBOL, at the "high-intermediate" level. It incorporates a facility similar to IBM VSAM, as well as interfacing to DBMS-X.

RPG II is based on IBM System/3 RPG II, together with features offered in other versions of RPG.

BASIC-X is a highly extended version of BASIC developed for business applications, providing an incremental compiler approach to program development and execution. The BTI 8000 BASIC-X facility accepts, without change, programs developed in BASIC-X on the BTI 4000 series systems.

Utilities and assembler:

The utilities package includes a versatile linking loader; an editor with a screen-oriented mode, for interactive development of programs, as well as for general document preparation; a general and powerful sort/merge which can be run conversationally, as part of a job stream, or as a sub-program; and a copy/format program for general-purpose data movement and conversion. Although BTI offers an extremely efficient single-pass assembler, PASCAL-X is strongly recommended as the preferred alternative in situations where assembly code might be used.

1.6 Management Services

The manager of a powerful multi-user computing system must have full control over the system and its use, but should not be burdened with duties which can safely be delegated to subordinates or automatically handled by the system itself.

In the BTI 8000, all system users are identified by accounts which reside in a management-defined hierarchy of up to four levels of control. All access to the system, including that of managers and operators, is via password-controlled entry to an account. The system manager can delegate supervisory authority to a series of division managers, who are allocated resources for distribution among their subordinate projects. Each division manager can in turn delegate authority to a series of project managers, who are allocated resources for distribution among their subordinate individual users. A manager (system, division, or project) can obtain comprehensive reports on the activities of his subordinates.

A powerful backup/recovery facility also contributes to control and convenience. Backup may be performed to any removeable medium, and is selective in several dimensions. Any account family (division, project, or individual account) may be backed up individually. Within each library, the backup procedure can automatically select files which have not been accessed (or modified, or backed up) since a given date, and will in all cases skip over files marked "no backup" by their owners. A variant of this procedure will back up and then purge from disk storage those files which have not been accessed for a specified time.

Accounts can be tailored for several specific purposes:

<u>Management</u>:

With proper authorization, an account can allocate and limit subordinate account access to system resources, and can obtain reports on subordinate account activity. The ability of an account to perform certain managerial functions does not always include the ability to pass these privileges on to other accounts.

<u>System operations</u>:

System operators need not and should not have the same privileges as management. The ability to adjust batch streams, operate tape drives and line printers, and perform system backup (without operator access to data) can be delegated to specific password-controlled accounts, and/or to execute-only programs within these accounts. All backup media are encrypted, for further security control. The system does not require a special operator's console.

<u>Public libraries</u>:

If not restricted by management, any account can declare any of the programs and files in its library to be public to another account or family of accounts. Accounts with certain predefined names have libraries which are implicitly public to their portions of the account hierarchy. Programs and files can be shared for common program access, common database access, or application team development.

Application development:

Programmers can be given individual private-library accounts for
on-line application development and testing, plus convenient access
to team development accounts. Program errors during development
and testing can be confined to one account library. Archival
library accounts can also be constructed for programmer convenience.

End-user turnkey operations:

With suitable applications design and the assignment of a startup
program to the account, turnkey end users can remain under the
full control of application packages for the entire duration of
every interactive session. End users, whether data entry clerks
or top management, need not be burdened with computer protocol
or jargon, and the application software need not be exposed to
misuse or harm.

General timesharing:

The default status of all ordinary user accounts is a two-way
isolation of activity and data from all other users on the system,
subject to the control of the account's manager. Three levels of
public libraries are available for the convenient dissemination
of programs and data, and system language facilities are available
for program development and use. The account's own library for
the storage of programs and files is private. Since all activity

can be monitored and controlled by supervisory accounts, the general user-account facility is ideally suited for training purposes, isolated development, and general timesharing. Because of system security provisions, the system manager even has the option of allocating a portion of in-house system resources for outside commercial timesharing; this can be done on a sublease basis, with some individual responsible for the management and control of the outside commercial division or project.

1.7 Failure Recovery

The BTI 8000 uses fail-soft design throughout. The operational aspect of this philosophy is that the consequences of a system failure should be minimized. The system should be immediately restartable, with minimum loss of operational continuity or data, and with as little staff effort as possible.

Power failure:

Recovery from power outages is automatic and does not require operator assistance; all processes resume where they left off. A built-in battery pack holds up the system real-time clock for power outages up to one week, so that system date and time are correct upon automatic restart. If power failure at the computer site disconnect telephone lines on dial-up modems, provisions can be made for safely and cleanly handling the remote processes affected, as discussed below.

Disk errors:

Historically, one of the most common causes for both data loss and system crashes is disk read error. Most systems, including the BTI 8000, can detect data errors: there may have been a "hit" on some random data bit, or there may be a series of erroneous bits, perhaps due to an electrical malfunction that occurred during their recording, or due to the deterioration of magnetic properties an a portion of disk surface. Two design facilities on the BTI 8000 essentially eliminate the problems caused by such errors. First, a unique disk data reconstruction mechanism, on which BTI has applied for patent, automatically regenerates the original data page from information present in a calculation segment added to the data segments that represent a page, even if one of those segments is totally unreadable. The page is then automatically rewritten in an alternate location, with the original location excluded from future access. Second, the Monitor records critical internal code and tables on the disk redundantly, providing a second level of safety.

Processor failure:

Since erroneous processor operation can destroy data, the system includes a large number of consistency-checking mechanisms to detect a failing processor. When the error is detected, the system halts, with the failure identified on the operator's display. In a multi-processor configuration, the system may be restarted as soon as the offending module is removed; no adjustments in user software are required to accommodate the temporarily reduced configuration, and the same job mix can be processed transparently to all applications at the temporarily decreased level of throughput.

Memory failure:

The system is tolerant of memory failures which are confined to a small portion of memory: the system strikes out the page containing the failure (notifying the operator), and continues without assistance. Gross failures of memory, however, have consequences similar to those of processor failure, and are handled in the same manner as processor failures.

Disk drive failure:

Since each account, its library, and the mass storage space that supports its use reside on some given "home" volume (rather than being split across volumes), loss of service on a disk drive (other than the one containing the system volume) affects only those processes which require use of the volume in question, and merely suspends the processes until the volume is brought back into service.

Line disconnect:

Normally, the majority of system use will be interactive terminal operations, often from remote sites on a communications link. If the system detects loss of the communications link on a terminal port while that port is active, it will close out the process(es) associated with that port according to user-programmed instructions. Line disconnect is an event (like keyboard interrupt) which can be "trapped" in user program code and handled by an interrupt service routine. Programmers can write service routines to take whatever actions are appropriate, including backing out partially completed transactions (although DBMS-X transactions do not require this), writing checkpoint information to files, etc.

System diagnosis:

System diagnosis is automated, at two levels. First, every major hardware module contains an independent microcode-controlled diagnostic section which exercises and diagnoses its module upon command from the operator's panel -- for example, whenever the restart key is pressed. Second, the BTI national service center uses a BTI 8000 computer to dial into all BTI 8000 systems in the field to perform automatic remote diagnosis on both a preventive maintenance basis (taking periodic "health checks"), and upon customer request. (Remote maintenance access can be disabled by a switch on the operator's panel, if desired.) Preventive diagnosis can reveal accumulated "soft errors" -- i.e., those which the system had been able to correct without assistance -- and indicate that a given component should be replaced, prior to its potential total failure. Twenty-four-hour on-call remote service provides computer-thorough fault diagnosis on an immediate basis, faster than a service engineer could arrive on the site. The factory service link is also the vehicle for timely and automatic installation of fixes and upgrades to BTI-supplied software.

2. SYSTEM HARDWARE: VARIABLE RESOURCE ARCHITECTURE

2.1 Engineering Design Principles

The concept of separation of function, introduced in Section 1, is specifically applied to engineering design in the BTI 8000. By analyzing functional requirements and then constructing function-specific hardware, BTI has been able to reduce hardware costs while increasing system performance.

Lower costs result from maximum utilization of components. By contrast, assemblies designed to provide multiple services through the same interface are always partially idle; the designer must ultimately compensate by introducing more assemblies. This is true both at the digital logic level and at the system architecture level, which separates the functions of program instruction execution (computation), memory access and control, and the management of data movement between memory and peripherals (channel I/O).

Higher performance results not only from the freedom to design the most efficient implementation of a specific function, but also from parallelism. If a multi-service assembly is faced with simultaneous requests for more than one of its functions, it must handle the requests one at a time. At the architectural level, a familiar example is the delaying of instruction execution by a traditional CPU while it processes I/O requests.

The BTI 8000 design takes advantage of current digital logic technology throughout its architecture. Intelligence is distributed: the major resource modules, and all peripheral controllers, are each special-purpose microprogrammed processors, which in turn make abundant use of microcomputer-based submodules for many of their service functions. Although the architectural approach is highly innovative, high-reliability proven technology is the only approach accepted for component selection and manufacturing-level design.

A further basic design principle is the commitment to automatic self- and cross-diagnosis and operational validity checking, throughout the architecture. Data parity is checked with every internal transfer, and, since horizontal microcode can take advantage of parallel operation capability, hardware that is not directly involved in an operation is put to use for self-consistency checking at every opportunity. At the highest level, every major module contains independent microcode diagnostics to exercise and check that module at system start time and upon operator request.

2.2 Bus Structure

The foundation of the architecture is the VRA bus, a distributed-logic, passive, synchronous bus with a 32-bit-wide data path (plus address, control, and four parity lines) and 16 slots for the attachment of major modules in priority order. All data transfers between major modules occur through the VRA bus at the rate of 66 2/3 nanoseconds per 32-bit word (60 megabytes per second), synchronized by a master clock residing in one of the modules, the System Services Unit (SSU), and by the bus driver and receiver circuitry that interfaces each module to the bus.

The four types of major modules are:

SSU	:	System Services Unit
CPU	:	Computational Processing Unit
MCU	:	Memory Control Unit
PPU	:	Peripheral Processing Unit

At least one of each type of major module must be present on the system. As discussed below, extra modules can be added up to the 16-slot capacity of the VRA bus. Only one SSU is required for any configuration; other units can be present in any combination, although functional tradeoffs in most applications will probably result in no more than eight of any one module type being used on a fully occupied bus backplane.

Physically, each module is a very large single printed-circuit board, mounted vertically, accessible from the front of the system cabinet behind a door under the operator's panel. The boards are metal-stiffened and furnished with oversized retractor clamps to facilitate easy insertion and removal by customer personnel, with no tools required. To reduce the risk of a single part failure incapacitating the system, the bus backplane itself consists of only passive conductors and connectors; bus intelligence is physically distributed to the interfaces carried on each module board.

2.3 SSU (System Services Unit)

Only one SSU is required for any configuration. Extra SSU's can in fact be plugged into the bus, however, as "hot spares", ready for immediate activation if the primary SSU fails.

The System Services Unit is a standard device with no optional characteristics or sub-units. It is a microprogrammed processor in its own right, and provides the following system services:

Master clock:

The master clock drives the VRA bus at 15 megahertz (one bus cycle every 66 2/3 nanoseconds). The clock pulse carried on the bus is also the source of synchronization for all units connected to the bus, and hence for the entire system.

Universal date/time clock:

This clock is the system's reference for wall-clock time and date, with a resolution of one millisecond. Following astronomers' conventions for universal time, it reports number of milliseconds since 0000 hours, 17 November, 1858, Greenwich time. (System software interprets this pure time into conventional date and time, allowing for time zone changes, etc.) A rechargeable battery unit, physically mounted on the SSU board, maintains the clock through power outages as long as one week. The clock is set at the factory, but can be adjusted through the Monitor; in this case (to minimize interference with software that references the clock) adjustments are applied gradually, at a rate no faster than one count in 10,000.

System ID number:

The SSU contains a special system identification number which is set at the factory and cannot be altered; this number is accessible to any program through a Monitor request. BTI systems are unique in allowing for the inviolate presence of third-party application software ("proprietary software feature"); by mutual arrangements between a system owner and a proprietary software vendor, third-party programs can be made inaccessible to even the system manager for purposes of listing or alteration. The system ID number provides a method for such proprietary software to ensure that it has not been improperly migrated to installations other than the ones its owner has authorized.

Operator's panel:

The SSU is internally cabled to the operator's panel, which is mounted at the top of the left-most system cabinet. The panel contains a plasma-display readout of ten alphanumeric characters (in blue) for reporting system status and exception conditions; a large, rectangular alarm light which flashes red to attract operator attention; and two rows of four rocker switches each, with a small green light to indicate the status of each switch. The lower row consists of the main power switch; a switch to disable BTI remote maintenance access; a switch to select between normal startup and dedicated diagnostic startup; and the run/halt ("start button") switch. The switches on the upper row, numbered one through four, are used to select from among sixteen possible variations of startup (normal warm start, normal cold start, etc.) or diagnostic operation.

Remote front panel:

The remote front panel facility in the SSU allows BTI remote maintenance access to the system to be capable of all diagnostic and control functions that could alternatively be performed from the operator's panel, including startup of a halted system. A cable connection between the SSU and the lowest-numbered Asynchronous Communications Controller provides the physical linkage for this access, and also identifies the active SSU if more than one SSU is present on the VRA bus. The lowest-numbered port on the system should be connected to a dial-in modem for remote maintenance purposes.

Intelligent bootstrap facility:

When the system is started, through either the operator's start switch or the remote front panel facility, the SSU first sends a signal through the VRA bus that instructs all system modules to run their built-in diagnostics. The results are reported back to the SSU, which in turn reports any abnormal conditions on the operator's display. The startup sequence also takes the roll of all system components: as the SSU initiates bootstrap of the Monitor into memory from mass storage, it adjusts the Monitor's internal system configuration table. Since the Monitor then adjusts its resource management operations according to this table, the end result is an automatic, one-pushbutton sysgen. The operator can reconfigure the system merely by adding or removing modules and then pushing the start button.

2.4 CPU (Computational Processing Unit)

In the BTI 8000, "CPU" means "Computational Processing Unit", as opposed to "Central Processing Unit": in a BTI 8000 multiprocessor (multi-CPU) configuration, there are no master-slave relationships among the processors, and hence no "center". A Computational Processing Unit also has functional responsibilities different from those of a conventional central processing unit.

Each CPU is a standard device with no sub-units cabled to it. The CPU is a 32-bit, general-purpose microprogrammed processor used for execution of program instructions. At any given time, each CPU on a system is running either a user process or a Monitor process.

Each CPU includes eight 32-bit program-accessible registers, all usable for general-purpose operation; a Program Counter register; a Process Status Register, containing condition bits and instruction trap controls; and a Monitor Status Register, containing internal interrupt control bits, a user/Monitor mode switch, and other operational information about the process and CPU.

Another important part of each CPU is the "page file", which is used to process every memory reference. Each element of the page file corresponds to one of the 128 pages of virtual memory that comprise the address space of the current process. The page file element contains the location of the physical memory page that the Monitor has assigned to represent that virtual page, if the virtual page is resident. Each element also contains access control bits, whereby a page can be marked as read-only or totally excluded, and access status bits, which inform the Monitor's scheduling routines whether a resident page has been altered (or referenced at all). The page file reference time is only one bus cycle (66 2/3 nanoseconds).

Machine instructions are all one word long, and reside on memory word boundaries. Operands, either in memory or in the registers, may be single 32-bit words, double words, or bit fields of one to 32 bits, including 8-bit characters, whose addressing is optimized by special portions of the CPU architecture and instruction set. There are 175 machine instructions available in user mode (as opposed to Monitor mode). The lower 22 bits of most instructions specifies an operand, with the three next higher bits sometimes used to specify a register. Many different ways of referencing operands are provided by the "address mode" field; indirect addressing further involves special one-word structures called "pointers", which themselves contain address modes and parameters for operand specification.

Most programs executed on the system are normally either Monitor code, or code generated by a language compiler (as opposed to assembler-generated code), so the data structures that are facilitated by these address modes are those most often used by compilers and operating system designers. The instructions and address modes hence facilitate manipulation of stacks and queues; arrays, with traps for bounds violation; and linked lists, all with arbitrary-size data elements. Character-string and bit-field manipulation is also performed by the machine, and there is a special set of instructions for subroutine linkage, with parameters passed by address or by value; traps are generated in the case of parameter mis-match.

A brief summary of CPU instructions and address modes is given in Appendix A.

2.5 MCU (Memory Control Unit)

Each MCU is a special-purpose microprogrammed processor which manages up to 16 megabytes of main memory. The memory on a given MCU must be consistent in type, although different MCU's can control different types of memory; thus future memory offerings can coexist on the same BTI 8000 with currently offered memory.

Core memory modules, cabled to the MCU, are available in 32 Kword (128 Kb) increments, with a minimum of 64 Kwords (256 Kb) present on a system. The 22-bit word address within the MCU allows a single MCU to control up to 4 million words (16 Mb) of memory. Each MCU-controlled core memory subsystem requires private power supplies; there are two sizes available, one to power up to 64 Kwords (256 Kb), the other to power up to 128 Kwords (512 Kb).

Core memory is non-volatile, and hence does not require battery holdup power. Full cycle time is 670 nanoseconds per 32-bit word, measured at the VRA bus interface of the MCU. Parity checking, by byte, is standard.

The system treats all physical memory, across MCU's, as a single resource with a continuous physical address space, managing this resource on a page basis; a page is 1 Kword (4Kb). The lowest 5+n pages on the system, where "n" is the number of CPU's, are reserved for resident Monitor use; the rest of memory is available for demand-paged placement of virtual pages belonging to user processes or the Monitor. If an MCU detects an error anywhere except in these low pages, the system can "strike out" (ignore) the page containing the bad area, and essentially reconfigure itself.

2.6 PPU (Peripheral Processing Unit)

Each PPU is a special-purpose microprogrammed processor which contains and controls four independent high-speed I/O data channels. Each channel supplies an 8-bit (plus parity) data path between the VRA bus and the device/communications controller to which it is cabled. Channels perform 8-to-32 and 32-to-8 bit blocking and deblocking of data, and engage in control dialogs with the subordinate controllers. Each channel interfaces to one controller, and includes internal buffer control intelligence to keep the data flow moving at full speed.

The PPU's are microprogrammed to perform channel management, including exception condition handling and data validation, with minimum CPU support. Once a CPU has supplied a PPU with the necessary parameters for a transfer, the PPU assumes full responsibility for performing the transfer between the controller and main memory, through the VRA bus.

Two of the four channels on each PPU are standard-bandwidth 5 megahertz data paths, capable of supporting all devices except mass storage. The other two are double-bandwith 10 megahertz data paths capable of supporting all devices including the storage module drive disk subsystems.

2.7 Peripheral Controllers and Subsystems

Mass Storage:

Each mass storage controller interfaces to the system through a double-bandwidth (10 megahertz) PPU channel, and can control as many as eight storage module disk drives.

The mass storage controller is an intelligent, microprogrammed device. It overlaps seeks on its subordinate drives, so that some stage of the disk access process can be in progress on all drives simultaneously.

The controller is also responsible for a unique formatting and error recovery facility. All disk transfers to and from memory occur one full page (1 Kword) at a time. When a page (along with its structural linking tag) is written to disk, the controller partitions the data into a number of segments, each of which is written out with synchronization, error-detection, and date-stamping information added by the controller. As this occurs, the controller dynamically constructs a block check segment calculated on the basis of the original data, and appends this segment to the others to complete the disk "block". If any segment is found to be erroneous when a block is later read, the original page of data (plus structural tag) can be dynamically and transparently reconstructed from the remaining segments. The block is then rewritten elsewhere on the disk volume, with appropriate substitution tables updated. This capability is totally unique to the BTI 8000, and BTI has applied for patents on its mechanisms.

Storage module disk drives are free-standing, removeable-pack units which are available in several capacities. Different sizes of drives may be mixed under the same controller. Access times are identical for all sizes, as follows:

Average seek time:	30 milliseconds
Average rotational latency:	8 1/3 milliseconds (3600 RPM)
Data transfer rate:	10 megahertz bit rate (1.25 Mbytes/second)

Storage module drive sizes are customarily given in "raw" byte capacities, which do not allow for formatting; standard unformatted sizes are 40 Mb, 80 Mb, 150 Mb, and 300 Mb. BTI prefers to describe these units in terms of usable data capacities, and thus offers drives as follows: 33 Mb, 66 Mb, 126 Mb, and 252 Mb.

Interactive terminals and asynchronous communications:

Each ACC (Asynchronous Communications Controller) provides up to 64 "ports" for the connection of bit-serial, RS-232-C, asynchronous devices, including interactive terminals and data communications modems. Ports are available in banks of eight ports each, with up to eight banks per ACC.

The ACC is an intelligent microprogrammed device which includes enough internal buffer memory to support simultaneous operation of all 64 ports at their maximum rate of 19,200 baud, with full-screen (1920-character) block-mode input from buffered terminals.

Through Monitor requests, programs can specify these operating characteristics, among others, at individual ports:

- o Adjust data rates to any standard rate from 110 baud to 19,200 baud;
- o Specify terminal interface protocol or (full-duplex) modem protocol;
- o Control request-to-send signals to interface with half-duplex asynchronous modems;
- o Enable or disable automatic echoing of received characters;
- o Specify any set of line terminating characters, so that, for example, "character-grabbing" word-processing programs can process individual characters as they arrive, or so that special end-of-message protocols can be honored;
- o Control separate input and output buffers, so that typeahead can be programmed if desired.

A "virtual terminal" software system is also provided, so that screen-formatting programs may be made independent of the precise characteristics of a given model of interactive terminal.

Other peripherals:

Controllers are available for two types of line printers; all printing is automatically spooled from disk under operator control. The medium-duty line printer runs at 300 lines per minute, with 132 columns of print. The heavy-duty line printers are available in 300, 600, and 900 lpm versions, all with 136 columns of print and vertical format control.

Nine-track magnetic tape controllers can control up to four tape drives each. Nine-track tape is industry standard (IBM/ANSI) halfinch, switch-selectable 800 bpi (NRZI) or 1600 bpi (PE), at 45 ips, with 2400 foot reels. These are real-time devices; that is, they are not spooled by the Monitor. They may be used for mass storage backup, in which case backup data is automatically encrypted, or for industry-standard inter-system communication with character data.

A 3M-type high-density cartridge tape capability is also offered for mass storage backup. The cartridge tape controller can support up to four cartridge tape drives; each cartridge can record up to ten megabytes of data.

2.8 Configurations

The unique flexibility of the BTI 8000's Variable Resource Architecture permits configuration plans to involve the three considerations of capacity, performance, and redundancy. Configuration plans should be analyzed and approved by BTI technical personnel.

The VRA bus resolves any occasional contention situations on a positional priority basis; bus priority (high to low) runs from left to right on the bus backplane. Memory access should be given highest priority, followed by channel I/O, then computation, and finally system services. Therefore the major resource modules should be installed in left-to-right order as: MCU's, PPU's, CPU's, SSU. There is no requirement to install the boards beginning with the leftmost of the 16 slots, and bus slots may be unoccupied between boards.

Configuration plans normally begin with the number of simultaneous users (counting a batch stream as the equivalent of several interactive ports, since there are no "ponder delays"); a description of the kind of work done by these users, which offers a rough guide to average program size and disk access frequency; and the amount of on-line file storage required.

A single CPU provides processing power in the "supermini" range. With suitable memory and disk configurations, a single-CPU system can effectively support between 16 and 64 users, depending upon the type of job mix and what kind of response time is acceptable. Total computational power on a multi-CPU system should be figured as a multiple of that provided by one CPU, less approximately ten percent for each additional CPU due to bus contention.

There should normally be at least as many MCU's as CPU's on the VRA bus; access to memory is a critical factor in a demand-paging system. Whenever possible, memory should be split across multiple MCU's , and the split should attempt to equalize memory capacity per MCU. Even in a single-CPU system, dividing memory across two MCU's allows contention-free PPU access to one memory bank while the other is used for program processing by the CPU; furthermore, character-move instructions can operate at essentially twice their speed if source and destination areas reside in different memory banks.

Physical memory requirements can vary quite widely depending on load/performance considerations. For configuration purposes, memory is measured in pages (1 page = 1 Kword = 1024 words = 4 Kb). A rough rule of thumb is to allow five pages for Monitor overhead, plus one page per CPU on the system, plus four to five pages per simultaneous user, assuming "average" kinds of transaction processing and development loads. Thus a system with one CPU and two MCU's, each with 64 pages of memory, should provide excellent response time for 24 simultaneous users.

In small systems (where there are empty bus slots), recommended practice is to plan on enough MCU's to limit memory per MCU to 64 or 96 pages (256 Kb or 384 Kb).

Access to mass storage (disk) is the other critical performance factor in a demand-paging system. In most situations, contention for a disk controller among its subordinate drives does not become significant with three or fewer drives per controller; if system load is relatively light, and/or the drives are used primarily for semi-archival storage of large databases, having more drives per controller should not seriously affect performance.

Contention among users for access to a disk drive deserves serious attention, since memory paging support of a user's process uses the disk volume on which his account resides. Limits on the number of users that can reasonably be supported on one drive vary from 8 to 32, depending on job mix, program size, and the amount of main memory on the system. In general, system performance will be improved by apportioning mass storage across as many disk drives as possible.

In planning the total amount of mass storage required on a system, one should allow for a generous amount of "free storage" as well as dedicated file space. Blocks of free storage are used for process support, including up to one-half megabyte of virtual memory per process; Monitor operations and tables; user scratch files; and spool files. In addition, four megabytes should be allowed for BTI software, Monitor, and maintenance areas.

Contention is not a significant factor in configuring the Asynchronous Communications Controller or other peripheral subsystems. If a redundant system is desired, however, then there should be enough unconnected PPU channels and controller ports to accommodate recabling of all equipment into a new configuration if a PPU or controller fails.

Fail-soft redundancy is available at the VRA bus level with the installation of multiple major resource modules (CPU, MCU, PPU). A second SSU may be installed to serve as a "hot spare"; it has no system function until recabled, but will be kept electrically powered to assure correct operation if it is ever required.

3. THE OPERATING SYSTEM: VIRTUAL MACHINE MULTIPROCESSING

3.1 Orientation

The BTI 8000 operating system, or "Monitor", pools and coordinates physical machine resources, including processors, to provide a secure, efficient, and convenient environment for multiple users of the system. The Monitor shields all user operations from actual hardware configurations, creating a standard, well-defined virtual machine for each user process. Since Monitor functions are so closely integrated with system architecture, the Monitor itself is protected from violation by user processes.

Since the Monitor is also responsible for automating as much of system operation as possible, the following discussion is presented for information only; internal functions need not concern the system owner, operator, or user.

3.2 Resource Management

Processors:

When the system is started, either from the operator's panel or through the remote maintenance facility, the SSU sends a start signal through the VRA bus, causing all units to run their self-contained diagnostics. Upon successful completion of this stage of system start, the first CPU to become ready temporarily takes over the system, locking out other CPU's so that it can control system initialization, reading in "resident Monitor" code from a known location on the system disk volume into the low pages of physical memory, and then executing that code. This is the only circumstance in which any CPU assumes dominance within the system.

When the other CPU's are unlocked, the system immediately enters its normal run mode. To begin with, there are no users on the system (assuming a cold start), and so all CPU's run that portion of Monitor code (from a fixed physical memory location) which investigates a task assignment table elsewhere in core; at this point there will be no tasks, so all CPU's will go idle. When a device (in particular, an ACC) signals the beginning of what might be a user log-on activity, the associated PPU places an interrupt signal on the VRA bus. The first CPU to respond will handle the interrupt, posting to the appropriate Monitor tables.

In the steady state of system operation, when there are more processes than processors, each CPU requests an interrupt from the SSU one hundred milliseconds after it "switches in" to any task. If there is no other reason why that CPU should not continue working on its current assignment (such as a wait-for-I/O roadblock), the interrupt will cause it to "switch out" after that much processing, and to again run the Monitor code that reassigns it to the next most eligible process.

The core tables used to direct and coordinate the activities of multiple CPU's are read and updated using software lockout. The lockout algorithms, and the CPU instructions used to implement them, are the same as those used in non-Monitor software to coordinate any set of cooperating simultaneous processes. A given (public) memory location is chosen, by mutual agreement, to contain a "lock" word. Before proceeding through a critical region of code (to be entered and executed completely by only one process at a time), the process executes a noninterruptible instruction which sets some special "locked" value into the public lock word, while simultaneously bringing the previous value of that word into private storage for examination. (The Set-and-Test, or Exchange instructions can be used for this purpose.) If the retrieved value is other than "locked", the process continues through the critical region, unlocking it when done. If, on the other hand, the retrieved value is "locked", then the process waits, since this indicates that some other process has entered the critical region.

Memory:

Even though memory modules may be physically interfaced through separate MCU's, the system treats all of memory as a single continuous resource. The low 5+n pages, where "n" is the number of resident CPU's, are unavailable for paging, since they contain resident Monitor code and tables. The rest of memory is used on a page basis for temporary location of code and data transferred in from mass storage, with no pre-assigned boundaries or regions.

When a Monitor routine executing in a CPU instructs a PPU to transfer a page into memory from mass storage, the PPU is given two memory addresses. One is the location of the page itself; the other is the address of a Monitor table element for storage of the structural information included in every mass storage block. In this way, programs can make use of the full 1024 words in every page, since pointers, flags, and other maintenance information are kept externally to the page contents. An analogous procedure is used on a write to disk.

The Monitor keeps track of the logical status of all pages in memory, including their "home" addresses on mass storage. Thus if a user requests execution of a program, the Monitor will search its lists before posting a disk read request, and will take advantage of the prior residency of any of the program pages to avoid disk access; any number of users may share any number of pages. This list searching takes place with every page-read request, including those for file data blocks; thus even file data pages are shared, to minimize overall system disk access. Access control flags associated with each page indicate whether the page is read-only or writable, and, if writable, whether it has been altered during its residency. This information allows pages of writable program data or file data to be shared among multiple users: they will share the same physical memory page initially, but the Monitor will create a private copy of a shared writable page for any process that issues an instruction that would alter the page contents, at execution time.

The access control and status flags, including a "page referenced" flag, are carried into the page files of the CPU's, so that the system need not make an extra memory reference merely to update or examine them. The "page referenced" flag is used to identify the working set of a process as it executes, for scheduling purposes.

<u>Mass storage</u>:

The system is disk-based, in the sense that structural information and operating parameters are ultimately entrusted to mass storage. Main memory is treated as a temporary area for process operation, with any structural or parameter changes written to disk; system restart presumes no information in memory. Thus the system's main concern in mass storage management is maintaining the integrity of its structures.

Disk drives, disk packs (which are removeable), and disk volumes (the logical contents of packs) are all identified separately, so that, for example, volumes can be copied from pack to pack (pack-private information includes "bad tracks" tables, which are associated with physical packs only). Files and libraries of files reside on individual mass storage volumes, so that volumes may be dismounted either logically or physically without halting system operation or destroying the integrity of structures. (The "system volume", containing the Monitor's operational tables and routines, as well as other data, cannot be dismounted.)

Internal system tables which are critical to operation, or to the use of an entire volume, are recorded redundantly in the interests of protecting operations and data. A general principle used by the Monitor is that during a structural update, the more junior table is created first and removed last. Even relatively complex structures are handled in a crash-resistant manner by using the (worst-case) technique of creating an entirely new structure (containing the new information and a copy of any previous information to be retained), updating the block that points to it, and finally freeing the old structure.

3.3 Process Management

A "process" is the distinct invocation, or separate execution, of a program. Each process on the system is usually, but not always, associated one-for-one with an on-line interactive user. (An interactive user process may generate other, "concurrent" processes; programs executed from batch queues are processes; and invocations of Monitor routines are processes.) Since the purpose of the BTI 8000 is to support many simultaneous processes (in fact, hundreds), it is properly described as a multiprocessing system, as well as a multiprocessor system.

The environment in which a process runs, and therefore the one for which programs are written, is its virtual machine. The monitor creates a basically private (but identical) virtual machine for each process; one of its aspects is the process address space, or virtual memory. Any and every program on the BTI 8000 may be written to address a continuous virtual memory of 128 pages (512 Kb), as if it were the only program executing on a private computer with that much physical memory. When a process addresses word zero, for example, it refers to address zero of its own virtual memory, not address zero of physical memory. The Monitor creates and maintains the correspondence between each page of every process' virtual memory, and some page in physical memory; this is what is loaded into a CPU's page file when a CPU runs a process. (Of a 17-bit direct address, the high seven bits select one of the 128 virtual pages and hence one of the 128 page-file elements; the lower ten bits then select one of the 1024 words within that page.)

Processes on the BTI 8000 may be running, in some CPU; runnable, but waiting for a CPU to become available; or waiting for some other resource, including a page of their virtual memory which is not yet resident in physical memory. Every process has all of its required virtual pages represented on blocks of a mass storage volume, but normally not all of them will be represented in physical main memory. When a running process references a virtual page that is not resident (as indicated by the page file), the process becomes suspended and the Monitor assumes the responsibility of loading the page from mass storage into some page of physical main memory. Since process pages are not brought into memory unless and until "demanded" by a process instruction reference, this is known as demand paging.

There are three characteristics of this technique that should be noted. First, the pages of a given process may be placed anywhere in paged memory. Second, a memory page that the Monitor chooses to overlay with a new page need not first be written back to disk, if it has not been altered since it was originally loaded in from disk (since a perfect disk copy already exists). Third, frequently referenced pages, including pages referenced by more than one process, tend to remain resident, since the Monitor's replacement algorithm tries to minimize disk access.

Demand-paging systems normally operate with a Least-Recently-Used (LRU) replacement algorithm; that is, the page chosen for overlaying is that which has "aged" the longest since being referenced by any process. This algorithm is entirely reasonable with a moderate load on a system, but invites a phenomenon known as "thrashing" when the load grows too large, as follows:

The pages used by a given process during a specified period of time are its "working set" for that period. In a demand-paging system, the relationship between the total pages required to hold all active working sets, and the total number of memory pages available, determines the amount of disk activity on the system. As the relative page load grows, disk transfers become more frequent, since executing processes more frequently require pages which have been overlaid by pages demanded earlier by other processes. At a certain point, all processes are reduced to their minimum working sets, below which they are incapable of executing any instructions without demanding a new page. Thereafter, any increase in load causes the system to spend almost all of its time in disk transfers, since every process is roadblocked by lack of a usable working set. In this "thrashing" state, essentially no work is performed.

The BTI 8000 modifies the conventional demand-paging algorithm to prevent thrashing. When a demand occurs, the Monitor selects the "least valuable process", based on a number of criteria (including the distinction between interactive and batch processes), and then strips this process of the least recently used page of its working set, overlaying that memory page with the one demanded. As the overall load grows, this procedure is repeated until all processes are reduced to working sets close to minimum. At this stage, prior to the thrash point, the Monitor identifies that process which is the most critical "troublemaker" -- normally, the one with the largest current working set. The Monitor then suspends this process for a certain period of time, rolling out its entire working set to free up memory for the rest of the load. The BTI 8000 process management algorithm avoids thrashing, therefore, by making a dynamic transition from demand paging to a modified multiprogramming technique.

There are no pre-set, conventional priorities in this scheduling technique. The Monitor automatically favors processes which are (currently) interactive, on the grounds that a user at a terminal requires service as soon as possible after entering a message. Processes which are not currently interactive can relinquish their demands for system resources (including CPU's) in favor of the former type of process, although a fairness algorithm ensures that batch processes are not totally locked out of execution. Process scheduling operates with dynamic priorities, according to the recent behavior and current characteristics of the processes. (One observation, however, is that on a heavily loaded system, with all other considerations equal, the most efficiently written programs -- those with compact working sets -- will be favored for execution over potential "troublemakers".)

4. USING THE SYSTEM

4.1 Account Structure

The system includes a four-level hierarchy, or tree structure, of accounts. This structure has two purposes: It organizes work done on the system, enabling managers to control the work of their subordinates; and it provides a framework for the system to ensure legal usage of system resources, while accounting for that usage.

An account is a logical grouping of all information concerned with an individual's use of the system, including programs and files that "belong" to that individual. An account should normally be provided for each authorized user of the system. An individual may have access to more than one account, and more than one individual may use the same account, even at the same time. The system, however, treats each separate account as if it belonged to a different individual.

Each account is identified by a name in the following format:

<u>username.dddppp</u> , where:

- o "username" can be up to 12 characters
- o "ddd" is a three-character division name
- o "ppp" is a project name of up to three characters.

The highest level of organization after the system itself is the "division". Certain predefined divisions exist:

o	.SYS	for system owner and operator control
o	.!!!	for BTI-supplied software, and maintenance access
o	.-xx	for proprietary accounts (defined by BTI)

The next level of organization (within division) is the "project"; individual user accounts then reside within projects.

Two usernames have special meanings: "MASTER" and null (no characters). The following illustration of account names shows how these are used:

MASTER.SYS	is used by the system manager, who creates and controls divisions, allocating resources to them.
.SYS	is an account whose library is public to (implicitly shared with) the entire account hierarchy; it contains programs and files supplied by the system owner for use by all users.
.!!!	is an account whose library is also public to the entire account hierarchy; it contains BTI-supplied compilers, utilities, etc.
MASTER.ddd	represents the accounts used by division managers, who create and control projects, allocating division resources to them.
.ddd	represents the accounts whose libraries are public to all accounts within their specified divisions.
MASTER.dddppp	represents the accounts used by project managers within their specified divisions, who create and control user accounts, allocating project resources.

.dddppp	represents the accounts whose libraries are public to all accounts within their specified projects.
anyname.dddppp	represents an ordinary user account, subject to the control of its project manager, its division manager, and finally the system manager.

Every account has certain properties, privileges, and restrictions associated with it. These include:

Passwords:

Access to each account, and hence to the system, is controlled by a password, which is a string of characters. Access is denied if the user fails to provide the proper password. An account may have a "future" password as well as a current password; the former replaces the latter automatically at a specified date.

Account library:

Every account may possess a private library of files, including program files. As discussed later, individual files in a library may be made accessible to other accounts, with various kinds of access restrictions. Files in public library accounts are implicitly accessible to other accounts within their family group in the account hierarchy. A "directory" is the internal file whose elements describe library entries; a "catalog" is a display of directory information. Each account library resides on some given volume of disk storage, each of which can accomodate up to 1024 distinct account libraries.

Hello programs and bye program:

When a user successfully gains access to an account, the system may automatically run a program known as the "Hello Program" for that account; this program is specified by the account owner or by an appropriate supervisory account. Hello Programs may be any user programs.

Two Hello Programs may be specified, in which case they are run one after the other. This allows the first to change frequently to provide system schedules or messages, while the second can remain unchanged to provide other services specific to that account.

Hello Programs may be designed to place the interactive user in a turnkey environment which controls and limits his interaction with the system, for specially designed end-user operation.

A "Bye Program" may also be specified to run when a user leaves an account environment, to properly close out his session by performing specialized accounting and posting functions.

Limits:

Each account has a set of limits for its use of various system resources. These limits are set by the appropriate supervisory accounts:

CPU time: This limit controls the total CPU time that an account may use; it is cumulative over all sessions. In addition to this cumulative limit, each account may have a single session limit, which may be expanded as needed by the user.

Wall clock time: This limit controls the cumulative amount of time that an interactive user may occupy a port.

Saved file block guarantee: This is the number of mass storage blocks reserved for this account. The user is guaranteed at least this many blocks for storage of files in his account library.

Saved file blocks limit: The account may not own more than this number of blocks of file storage under any circumstances.

Saved file blocks warning limit: If a user owns more than this many blocks in his account library, he will receive warnings from any system utility that increases the size of his library. (This is to help prevent surprise program terminations.)

Scratch file block limit: This is the total number of "unsaved" (temporary) mass storage blocks that an account may intantaneously own. If more than one user process is simultaneously running under the account, this limit applies to total use.

4.2 Account Usage

System control:

The system distinguishes between management control and operator functions. System operator accounts may be created within division .SYS to control batch streams and coordinate use of devices such as tape drives and line printers; the authority to manage other activities is held by the system manager, his division managers, and their project managers.

For example, if a manufacturing company purchases a BTI 8000, the person in charge of the system might decide that there are basically three divisions that will be using the computer: accounting, given the division name "ACC"; shipping, given the division name "SHP"; and production, given the division name "PRD". The system manager, using the account MASTER.SYS, then creates the account MASTER.ACC with sufficient resources to do its job, and gives the account password to the person in the accounting department designated as manager of its computer usage. Similar accounts are distributed to shipping (MASTER.SHP) and production (MASTER.PRD).

The manager of the accounting division might then decide that there are several projects within his division: development of new accounting programs (ACCDEV), the auditing project (ACCAUD), and accounting report generation (ACCREP). For the development project, for example, the accounting division manager, using the account MASTER.ACC, creates the account MASTER.ACCDEV and gives its password to the person selected to be manager of that project. He in turn will create accounts to be used by individual programmers (JONES.ACCDEV, SMITH.ACCDEV, etc.) and/or accounts to work on certain tasks within the project (FORECAST.ACCDEV, etc.).

Managers at all levels can obtain comprehensive reports of computer usage broken down by individual subsidiary accounts or tabulated by project (for division managers) or by division (for the system manager).

Program development:

Individual programmer accounts may be created for application program development. These accounts should have non-restrictive Hello Programs, such that the user will find himself in the system's "Control Mode", an interactive environment that allows him to run editors, compilers, debuggers, and other program development tools, to test his programs, and to exercise control over his account library. The programmer also automatically has access to files and software in the project, division, and system public libraries.

The system also allows the possibility of creating "project accounts" as well as "programmer accounts". Normally, if a running program or interactive user specifies the name of a file but does not specify which account library it resides in, the system assumes the file is to be found in the user's own library. A special system feature allows an alternate default library to be specified, so that this other library is searched if "owning account" is not named. Hence accounts can be created not only for individual programmers, but also for the major jobs they are working on; the latter accounts can be set up so that they are not accessible for normal interactive log-on, but serve instead as repositories of files related to some given job.

Thus in the above example, the programmer Jones could log onto his "home account" (JONES.ACCDEV) and, if he were working on the new forecasting system, could set his default directory to the account holding the work for that job (FORECAST.ACCDEV). This structure keeps programmers separate from projects; access can be given to or restricted from individual programmers through project manager control.

Turnkey user environments:

A typical use of the account structure is for turnkey end-user application. Any user account can be assigned a Hello Program which does not release its interactive user to Control Mode, but instead retains conversational control or passes control to another program directly. Programs can be made non-interruptible, and can assume charge of all aspects of interactive use. Programmers can set "traps" in such turnkey programs, so that data errors or program errors can be handled without end-user involvement.

An interactive turnkey end-user can thus begin a session knowing only how to log on with a specified account name and password. From that point until the end of the session, he need not be burdened with any protocols beyond those constructed by the application program designer. The application programs, in turn, are insulated from damage by any user action.

Turnkey transaction processing can be used for any purposes from low-level data entry to management report displays.

Proprietary software:

Third-party vendors of value-added software packages traditionally are faced with the problem of protecting their investments. If source code is released to the purchaser, the software is subject to local modification (which can severely affect the difficulty and cost of maintenance), and what should be proprietary and valuable invention is protected only by licensing or rental contracts. If only object code is released, maintenance becomes awkward, and the software is still vulnerable to unauthorized migration.

The BTI 8000 is unique in providing for an arrangement to solve these problems. A series of account names (division names beginning with "-") is reserved for creation and control by BTI only; accounts so named are called Proprietary Accounts. The system manager of a BTI 8000 (MASTER.SYS) can limit the amount of resources used by a Proprietary Account, and can remove the account entirely; he cannot, however, control the library of such an account, nor can he log into the account without knowing its password.

The vendor of a proprietary software package can arrange to have his software transported to his customer's system in the form of a Proprietary Account, whose password is known only to the software vendor. The latter can declare his programs executable but not readable by the users on his customer's system. Then, he can still access source code on the remote system (by logging in as an interactive user over a telephone line), to perform software maintenance conveniently.

As further protection for the software vendor, each BTI 8000 system contains a unique and fixed, but program-accessible, system ID number, so that programs can check to insure that they have not been improperly migrated to other systems.

General timesharing:

The security and controllability inherent in the account structure make the BTI 8000 an excellent system for commercial timesharing use, even as an adjunct to in-house use. A division or project can be "split off" for this purpose (or equivalently, for in-house casual or training use), without concern for the privacy or integrity of other activities on the system.

It is even possible to enforce fiduciary relationships in a timesharing situation: a supervisory account (project, division, or system manager) can formally relinquish its authority to control the account library of a subordinate account, and/or to control its account passwords (and hence to log into the account). Once relinquished, these authorities can be reestablished only by the action of the subordinate account.

E 4

4.3 Interactive Access

Each asynchronous communication port on the system can be used for an independent, interactive conversation with a process running under some account.

Local terminals (within 1,000 feet of the computer) can be connected directly by cable. Terminals at remote sites may use either public ("switched network") or private ("leased line") data communications facilities to reach the computer. Although groups of remote terminals can be multiplexed to reduce communication line costs, the individual data paths must be demultiplexed at the computer site, since each terminal must be logically connected to a separate port.

After establishing the connection between his terminal and an asynchronous communications port, the interactive user informs the system of the account name he wishes to use, and the password. The system compares the entered password against the currently valid stored password; if it is accepted, the user is considered logged on to that account, and the system runs the Hello Program(s), if any, for that account.

In high-security applications, the Hello Program(s) can be made non-interruptible and used to validate a "secondary password", which can involve an arbitrarily complex (and even randomly changing) dialog between the user and the program.

4.4 Batch Processing and Operator Functions

Batch processing in the BTI 8000 operates through spooled, virtual card readers called "task queues". Text records, representing (variable-length) card images, are placed in the task queues by processes which treat the task queues as system-owned sequential disk files. Any process may write records onto the task queues. Interactive processes may generate task queue records directly from keyboard data (in the manner of a direct keypunch replacement), from disk files, or from magnetic tape. The system operator may also start a process to build these records from data coming from a unit-record device or communications link.

A batch "job" is a sequence of task queue records that begins with an account log-on ("HELLO" to some valid account, plus password) and ends with a log-off, in the same fashion as an interactive session. The records in a batch job are logically equivalent to lines entered at a keyboard during an interactive session; the same "Control Mode" commands are used. (One difference is that the Control Mode "DO file" commands are valid in a batch job; see Section 7.1.)

A sequence of records submitted to a task queue as one unit is known as a "deck". A deck may consist of one or more jobs.

The system manager can configure the task queues. Separate task queues can be established according to several criteria: the amount of service each queue will receive (by arrangement between the system manager and the system operator); the time of day each queue will be serviced (as above); the number of non-spooled peripheral devices, especially magnetic tape, required by each job and deck in each individual queue (e.g., there may be a one-magtape queue, a two-magtape queue, etc.); or some combination of all of these criteria. Different accounting charges can be specified for the various task queues, each of which is identified by a unique name.

The system operator logs onto an account within division .SYS which contains an "operator's program"; this is a program to which the system manager has granted the privilege of running the special operations required to perform the system operator functions.

The system operator can balance system load by manipulation of the task queues. He can obtain a list of the various task queues and their defined attributes, and for each one, can:

- o inspect the sequence of decks, and the identities of the jobs within decks;

- o rearrange the sequence of decks within the queue;

- o instruct the Monitor to process the first "n" decks at the head of the queue; the operator can set "n" to any number, including zero (thereby shutting down that particular task queue).

The operator's program also displays a list of all processes active on the system, identifying each by a process ID number. For each process ID, the following information is available to the operator:

- o interactive port number, or task queue name
- o account ID
- o account resource limits
- o resources used in this session (including time on)

- o number of non-spooled devices (e.g., magtape) in use
- o name of currently executing program

The operator can ABORT, EVICT, or STOP (and later GO) any active process. An ABORTed process is destroyed, but its "line disconnect" interrupt routine, if any, is processed first; the effect is as if the process were an interactive process which detected telephone line disconnect. An EVICTed process is destroyed immediately. A STOPped process is suspended from execution, but may be reawakened later (with GO).

Besides controlling batch execution, the operator can also set a limit on the total number of log-on sessions to be simultaneously serviced by the system.

Another important operator function is the granting of requests for non-spooled peripheral devices, especially magnetic tapes. Such requests can arise from interactive as well as batch processes, and are handled as described in Section 6.

The operator is also responsible for setting default characteristics on interactive ports (baud rate, etc.), and for mounting and dismounting disk volumes (logically as well as physically).

The operator can start up processes on remote ports (ports other than the one he is using) to accommodate receive-only devices and other arrangements where no explicit log-on command is expected at a given port.

Finally, the operator can run the BTI-supplied BACKUP and PURGE utility programs, which select portions of the system to back up, and the backup medium. Backup can be selective in several dimensions, including portions of the account structure and various characteristics of files, as discussed later in this manual.

5. PROGRAM OPERATION

5.1 The Virtual Machine Environment

All running programs (processes) function in a standard virtual machine environment created and maintained by the Monitor. The name "virtual machine" is applied because the description of the process environment is conceptually equivalent to the description of a machine on which each process runs; processes are insulated from the "real" (hardware) machine.

The instruction set of the virtual machine is the "user-mode" instruction set of the Computational Processing Units, extended by a list of several hundred psuedo-instructions known as XREQ's (pronounced "ex-reks", meaning "executive request"). An XREQ pseudo-instruction looks like an ordinary CPU instruction, but its execution within a CPU results in that CPU "trapping out" to a Monitor routine to service the request. (When a CPU switches from user code to XREQ-service Monitor code, it does not store the entire process state, thereby saving switching time.) XREQ's are provided to handle all potentially sensitive operations, so that process and system integrity can be maintained; XREQ definitions also tend to be quite simple, so that the burden of executing fairly complex operations is passed from the user program to the Monitor, extending the power of the virtual machine.

The user process does not directly see the operation of the Monitor or the presence of other users. The only signs of the existence of the Monitor are the operation of the XREQ's and, since processes are scheduled "in" and "out" of processors, the inevitable difference between virtual time and real time.

The system's management of virtual memory is also invisible to the user process, which sees its 128-page (512 Kbyte) address space as continuous and always available. The Monitor moves the pages of this address space to and from the disk on demand ("demand" meaning actual reference to a page); a process is unaware of its suspension while a demanded page is moved in from the disk, as it is unaware of suspension for any other resource-roadblock reason.

The 128 pages of process address space form the directly usable memory of the virtual machine. When the system performs operations on behalf of a user process, it executes in another virtual address space (one separate from the user's). In this case, however, CPU time and other resources involved in XREQ execution are charged against the user making the request. The Monitor code that performs the XREQ's is swapped in the same manner as user pages, and thus is resident only when there is a demand for the particular system function. Pages moved in on this basis enter into the scheduling algorithms which "rate" processes on their need for memory, but tend to be shared by multiple users.

A process need not take a merely passive view of its address space; XREQ's are provided to "map in" pages of random access files and executable code files into virtual memory pages. When a page of a random access file, for example, is mapped into one of the 128 pages of process memory, the pages become equivalent: an instruction which reads or writes data from or to that page of the process address space actually accesses the specified page of the file.

The mapping facility can be used to manage program overlays; to perform direct I/O with random access files, as an alternative to using READ and WRITE XREQ's; and to facilitate inter-process communication (as an alternative to other techniques), by having several processes map in the same page of a multiple-write file. The overlay technique effectively expands program size to any limit desired, while the file-mapping technique allows a program to deal with arrays, for example, much larger than the one-half-megabyte limit of process address space.

In summary, the virtual machine in which each user process executes includes the following elements:

- CPU instruction set
- XREQ pseudo-instruction set
- Account identification (Every process runs under the auspices of some account, with its account library, privileges, and restrictions.)
- Machine registers (8 general-purpose, Program Counter, Process Status Register, Monitor Status Register)
- 128-page address space (512 Kbytes), with overlay and file-mapping capabilities
- I/O, file system, and inter-process communications facilities, controlled by XREQ's (discussed later)
- Process structure status, process authority status, and interrupt facilities (discussed later)

The instantaneous states of the last 5 of the above elements form a complete description of process state.

5.2 Concurrent Processes

The simplest view of user program execution is that of each interactive user (or batch job) running one program in the virtual machine associated with his communications port (or batch stream). Just as program execution capabilities can be expanded in physical machines by techniques such as multiprogramming and overlays, virtual machine capabilities can be expanded by similar techniques: concurrent processing and process structuring.

When an interactive user logs on at a port (or a batch job is started), a "primary process" begins. This process runs in the virtual machine process environment described earlier, and is associated with the user's port (or batch stream). Any process, however, can issue an XREQ to generate another process, which runs concurrently and asynchronously with the process which generated it. ("Spawning" is a term often used to refer to "generating" a process.) A process may generate a series of concurrent processes, which in turn may each generate a series of processes. Each process runs in its distinct virtual machine, but a family of concurrent processes may include only one (the primary, original process) which is in direct communication with the original port or batch stream. Concurrent processes communicate with one another through inter-process communication links known as .PATHs, described later in this manual; in particular, the spawning XREQ automatically creates a pair of these links between the senior and junior processes.

Concurrent processes find their utility in two ways. First, a time-consuming task which is amenable to division into asynchronous processes may be processed "in parallel" using multiple concurrent processes. Given a suitable number of CPU's on a system and a light enough external load, this kind of multitasking is true parallel processing; the pieces of the job will be processed in parallel, in the real-time sense.

Second, mutually "suspicious" programs can cooperate, with each knowing that the other cannot access its memory or files. A primary process can spawn a concurrent process and then, given permission to do so, load that virtual machine with a program belonging to another account to perform some sensitive function, such as extracting limited information from a protected database.

5.3 Process Structuring and Run Control

The other virtual machine expansion technique is the "structuring" of a single process to establish a program control order.

A program running in a process environment (virtual machine) can set up and then start an "underprogram". The underprogram takes over the virtual machine from the program that started it (its "overprogram"), but the system saves the latter's status information; when the underprogram stops, the system brings the overprogram back into the virtual machine and resumes its execution at the instruction immediately after the XREQ that originally started the underprogram.

Of the process environment (virtual machine) elements described earlier, the following are collectively known as "program console" items, and are kept local and distinct for each over/under program:

- Machine register contents (8 general-purpose, Program Counter, Process Status Register, Monitor Status Register)
- Address space (i.e., contents of the page file)
- Program interrupt mechanisms (discussed later)

The other elements of process status (in particular, linkages with the file system) remain unchanged as different over/under programs assume control.

Since an underprogram can in turn set up and start its own underprogram, processes can be structured into "layers" of over/under programs. Only one of these at a time, of course, will occupy the process registers, address space, and interrupt mechanisms, and be executing; a process status element, however, keeps track of the structure, and of all "program consoles" that are not currently in use.

When a process begins, the system establishes its "overmost" (and only) program automatically, and begins its execution: this is the BTI-supplied "Control Mode" program, which serves as the system's command language/job control language interpreter for both interactive and batch processes. When a user requests execution of an application program (including a system language compiler or utility), or if a Hello Program is indicated, the Control Mode program runs that program as its underprogram within the process environment.

The following XREQ's control over/under program structuring of a process:

- SETUP: After a program has been identified through file system linkage, this XREQ establishes that program as the underprogram; it can be issued only by the (currently) "undermost" program.

- GO: This causes the system to save the program console information of the current program, substitute that of the next underprogram, and then begin execution of the latter. Prior to issuing the GO the current program can manipulate the console registers of the underprogram, including its program counter (to specify a starting address).

- **STOP:** When a program executes a STOP XREQ, possession of the process environment reverts back to the immediate overprogram, with execution resumed just after the GO XREQ which started the underprogram. One of the overprogram's registers is loaded with a code which indicates the reason for the STOP (normal end, error condition, etc.).

- **INSERT:** As opposed to SETUP, this XREQ can be issued by a program which already has a program below it in the process structure. The effect is to "pry apart" the layers and insert the selected program between the current program and what had been its immediate underprogram. It is used chiefly by debuggers, as described later.

- **STEP:** As opposed to GO, this XREQ brings in and executes the underprogram for the execution of a single instruction only, after which control reverts to the overprogram. (An XREQ is treated as a single instruction.) It is also used chiefly by debuggers.

- **DESTROY:** This removes the immediate underprogram from the process structure.

- **DESTROYALL:** This removes all underprograms from the process structure, leaving the current program as the "undermost" one.

- MAPIN, FREEPAGE: These XREQ's can be used to map (and then "unmap") a page of an underprogram's address space into the current address space; this allows the current program to examine and/or manipulate the underprogram's data areas before or after running that underprogram. Of course, the security status of the underprogram must be such that these actions are permitted.

Note that in all cases, it is the overprogram that manipulates the underprogram and its execution. Control authority applies downward, not upward, in the structure, and so one can say that overprograms control underprograms.

There are three main uses for the process structuring capabilities.

First, Control Mode itself, upon user request, runs a requested program as an underprogram. When the underprogram stops, the system returns control to the Control Mode program in a manner that allows Control Mode to conveniently ascertain why the program stopped, and to deal with errors, if appropriate.

Second, BTI-supplied language debuggers can be invoked after a compiled program has halted with an error. The program which ran the halted program (normally, Control Mode) merely uses the INSERT XREQ, placing the debugger between itself and the halted program, and then runs *that* debugger; the debugger program can examine and modify the compiled program (its underprogram), and then run it, perhaps even in single-step mode.

Third, the generality of process structuring allows programmers to construct their own "control mode" program, either to create a special-purpose interactive environment or merely to have a conversational design different from that of BTI Control Mode. The special-purpose "control mode" program is simply assigned, on an account-by-account basis, as the Hello Program. When a user logs on, Control Mode runs this program immediately, and the user then sees the interactive protocols designed for him. Such a special-purpose "control mode" program can run other programs in exactly the same way that BTI Control Mode can; furthermore, such a program can be "rescued" by BTI Control Mode in case of error, or merely if its designer decides to leave certain capabilities to BTI Control Mode.

5.4 Program Interrupts

User program interrupt facilities are bound to programs as opposed to virtual machines; they are part of the (local) program "console", not the (global) process status.

Interrupts begin as events which fall into one of several classes, including:

- o arithmetic fault (divide by zero, etc.)
- o misuse of CPU instruction set
- o timer notification (an XREQ had requested interruption after a specified number of milliseconds)
- o terminal BREAK key sensed (keyboard interrupt)
- o port disconnect (the Asynchronous Communication Controller sensed a voltage drop that normally indicates telephone line disconnect or terminal power-off)
- o non-empty .PATH (a message has arrived on an interprocess communication facility)

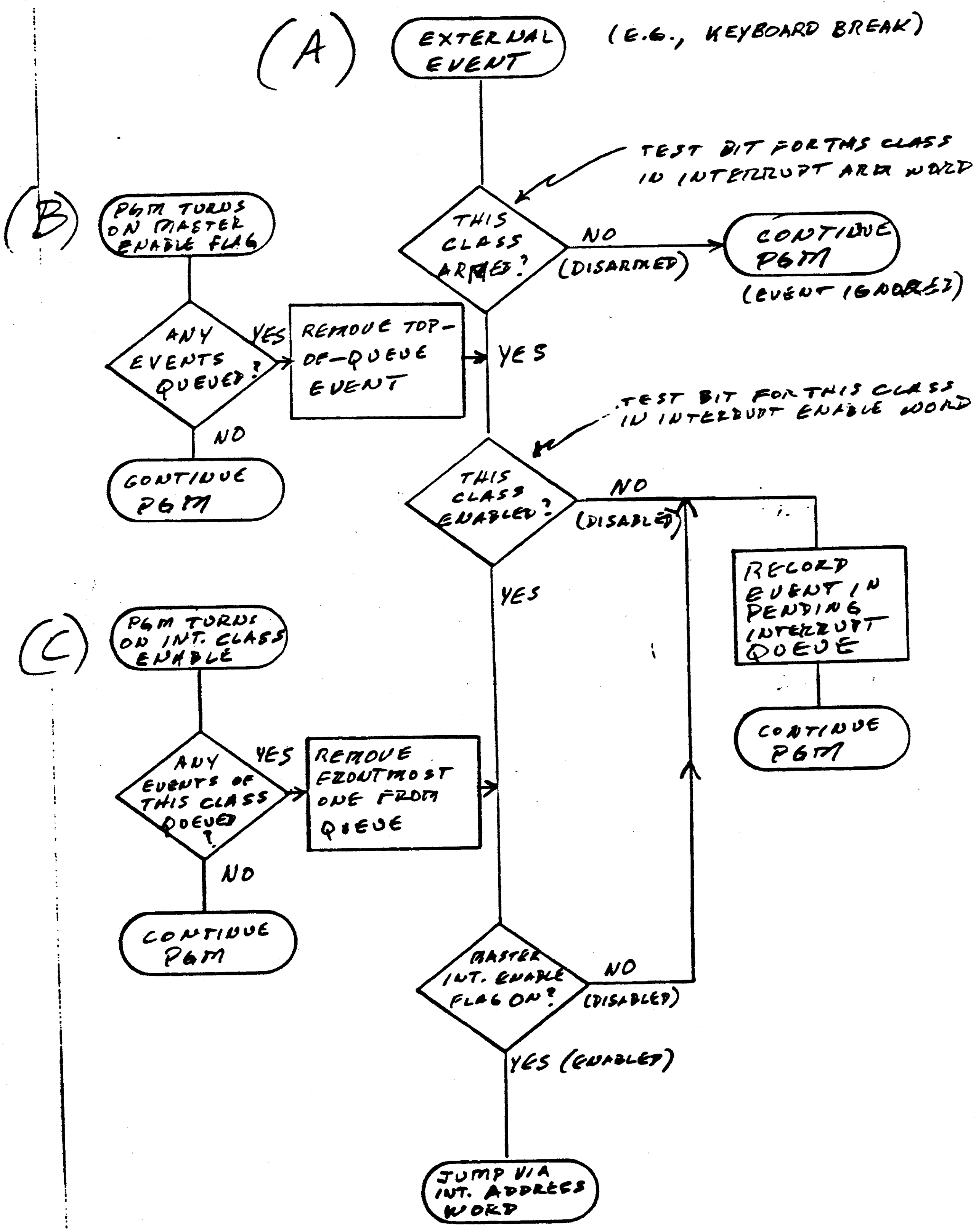
When such an event occurs, the system first examines the on/off status of a bit selected from the Interrupt Arm Word: each class of event is represented by a different bit position and the user program can manipulate and examine this word with XREQ's.

If the bit is off, that class of interrupts is said to be "disarmed", and the event is ignored. If the bit is on, the interrupt class is said to be "armed", and the system then examines the corresponding bit position in the Interrupt Enable Word, which is also under user program control.

If this bit is off, that class of interrupts is said to be "disabled", and information about the event is placed on a first-in-first-out Pending Interrupt Queue with no further action; this queue can be examined by the user program. If this bit is on, that class of interrupts is said to be "enabled", and the system proceeds to examine the (single) Master Interrupt Enable Flag. (If an event is queued and the appropriate Enable Word bit is later turned on, the system takes action as if this event had just occurred.)

If the flag is "off", the effect is the same as if the interrupt had been disabled at the Interrupt Enable Word. If the flag is "on", the system diverts program execution to the address given by the Interrupt Address Word, where the program presumably will begin its interrupt service routine.

These facilities provide for totally shutting out interrupts by disarming them (although CPU instruction errors can never be disarmed); queueing them for later service (armed but disabled); and conveniently postponing all servicing through the Master Interrupt Enable Flag (for example, when entering an interrupt service routine). Furthermore, the interrupt service routine can choose to allow the controlling overprogram to service the interrupt, merely by executing a STOP XREQ; most programs allow Control Mode to handle "unexpected" interrupt events.


(A)
EXTERNAL EVENT
(E.G., KEYBOARD BREAK)
TEST BIT FOR THIS CLASS IN INTERRUPT ARM WORD
THIS CLASS ARMED?
NO (DISARMED)
CONTINUE PGM (EVENT IGNORED)
YES
(B)
PGM TURNS ON MASTER ENABLE FLAG
ANY EVENTS QUEUED?
YES
REMOVE TOP-OF-QUEUE EVENT
NO
CONTINUE PGM
TEST BIT FOR THIS CLASS IN INTERRUPT ENABLE WORD
THIS CLASS ENABLED?
NO (DISABLED)
YES
RECORD EVENT IN PENDING INTERRUPT QUEUE
CONTINUE PGM
(C)
PGM TURNS ON INT. CLASS ENABLE
ANY EVENTS OF THIS CLASS QUEUED?
YES
REMOVE FRONTMOST ONE FROM QUEUE
NO
CONTINUE PGM
MASTER INT. ENABLE FLAG ON?
NO (DISABLED)
YES (ENABLED)
JUMP VIA INT. ADDRESS WORD

PROGRAM INTERRUPT HANDLING

6. PROCESS INPUT/OUTPUT

6.1 Design Principles

I/O design in the BTI 8000 uses the following principles:

- Provide a basis for building applications-oriented data structures on storage media.
- Give application programs complete control of the user interface, thus allowing error recovery and human engineering of applications.
- Ensure the integrity of data held by the system on data storage media.
- Enforce protection of data stored on various media (disk and tape).
- Provide as much device independence as possible.

6.2 Record Types

The system uses record-oriented (equivalently, line-oriented) I/O, as opposed to the "virtual terminal" approach, in which every I/O device would be modeled as an ASCII terminal. Most higher-level languages are record oriented, and it is more efficient to examine record headers for special treatment (such as positioning for new lines on output to a printing device) than it is to examine every character. On output devices, end of record is interpreted as end of line; on input, one input "line" is one record.

Records are strings of characters of a specified length which, to extend their utility, have a "type" associated with them to suggest an interpretation for using their contents. Record types are:

- Text: The contents of a text record are considered ASCII text characters with the end of record denoting the end of the line.

- Text with formatting: As above, except that the first character is considered as a special output device formatting control. When output to a printing device, the character is interpreted as a print format command.

  Traditionally, a line printer is a pre-print device (e.g., a space character causes the paper to advance before the line is printed), while terminals are usually post-print devices (a line is terminated by a carriage return - line feed pair, which repositions the carriage after the line is printed). The only real difference is in how overprinting is to be handled: for the line printer, each overprinting line save the first is marked, while for the terminal, each line is special save the last. This pre- and post-print difference is resolved by having the system remember how the last line was printed, and interpreting the format control character to mean how the line should appear on the page (as opposed to what specific actions should be taken before or after printing). Thus, a space causes that text line to be printed on the line after the last line printed.

- <u>Binary</u>: A "binary" record contains 8-bit bytes of coded information, with no interpretation imposed by the system.

- <u>Comment</u>: A plain text record embedded in text but not part of the information stream. Compilers that read them place them on the output listing (if one is being made), but do not process them as input.

- <u>Forms</u>: A plain text record that has meaning when output to some printing device. For example, when output to a line printer, a forms record causes the printer to stop, and the text in the record is printed out to the operator to request change of paper, etc.

- <u>Label</u>: A plain text record interpreted as a label to be placed on the printing output device (expanded to large letters on the line printer, etc.).

- <u>Filemark</u>: A special record that simulates the filemarks found on magnetic tapes. It can be written to sequential disk files or output devices (with interpretation depending on the device). It can be read from sequential disk files, magnetic tapes, and other sequential input devices. (A filemark cannot appear on a random access disk file.)

- End of Data: A pseudo-record type that signifies the physical end of the sequential input medium. It is most often found on sequential disk files when all the data in the file has been read.

- Abnormal End of Data: A pseudo-record type that signifies that there is no more data on this input device, but the reason is that the data on the file has been lost due to system (or disk) failure.

6.3 Virtual Channels

Each process owns two hundred two virtual I/O channels, through which all information into and out of the process passes. Each virtual channel may be attached (assigned) to some I/O medium, or logical device, including files. Virtual channels are numbered 1 to 202, and the term "lun" (pronounced "lunn", meaning Logical Unit Number) is often used synonymously with "virtual channel". Since all I/O and file operations are performed through the virtual channels, the term "lun" is also often used to refer to the device or file to which a virtual channel is attached, so that one can speak of rewinding a lun, reading a lun, etc.

The term used to describe the attachment of a virtual channel to some logical device or file (and the XREQ that performs the assignment) is "equip". Thus luns are "equipped" and "unequipped".

Some virtual channels have fixed or at least pre-specified assignments:

lun 1:	"Standard" process input.
lun 2:	"Standard" process output.
lun 200:	Equipped to the file containing the executable code image from which the current program was taken.
lun 201:	Fixed process input.
lun 202:	Fixed process output.

Luns 201 and 202 may not be unequipped; in an interactive process, 201 and 202 are the terminal, while in a batch process, lun 201 is the (spooled, virtual) card reader and lun 202 is the (spooled) line printer. When a process begins, luns 1 and 2 have the same assignments as luns 201 and 202, but they may be unequipped and reequipped to other devices. Thus if a program outputs tables, for example, on lun 2 so that they will appear on the line printer from batch, a terminal user who is debugging that program could equip lun 2 to a file and save the output instead of just letting it appear on the terminal; program data output would go to lun 2 while error messages and checkpoint notices would go to lun 202 (and appear on the terminal, in this case).

6.4 Logical Devices

Equipping a lun attaches that virtual channel to a logical device of one of several "device types". Each device type has well-defined attributes and access behavior. There are two classes of logical devices, grouped according to their type:

First, there are those which by their nature cannot be "owned" as an entity in any account library. In general, a process sees only one of each type of logical device in this class, and that logical device is usually a close analog to some physical device (so that each may be regarded as a virtual peripheral device connected to the user's virtual machine through a virtual channel). The following abbreviations are the formal device type names for devices in this class:

.TERM	the user terminal connected to an interactive process
.LP	spooled line printer (a process may equip to several line printer spool files -- i.e., several .LPs)
.MT	magnetic tape drive (several visible to a process)
.NULL	the system "bit bucket", or write-only memory
.TASK	the card-image writer's view of the virtual (spooled) card-reader input hopper for batch (several)
.CDR	the (batch process) card-image reader's view of .TASK
.DIR	the directory of an account library (a process may be allowed to see directories other than that of its own account)

Second, there are those logical devices which can be owned by an account, as entities in an account library. For each device type in this class, a process sees an infinity of available devices (instances of the device type). Some of them may have names (and other attributes more specific than those implied by type alone); these will also be owned by some account (especially the account to which the process belongs), and, if the device type is appropriate, may contain data. Others (conceptually, an infinity of them) have no names and no owners, and are empty. Roughly speaking, each device type in this class may be regarded as a type of file. These logical device types are:

.SAF	Sequential Access File
.RAF	Random Access File
.CODE	executable program memory image file
.PATH	inter-process communication link
.LOCK	inter-process event coordination semaphore

A logical device that has a name and an owner is one that has been "saved" in some account library. A process equips a virtual channel to such a device by specifying (with the EQUIP XREQ) the name of the account and the name given to the logical device; the device type need not be specified, since it is known as soon as the equip finds that device. Saved logical devices are commonly referred to as "saved files", although .PATHs and .LOCKs are not truly files.

Alternatively, if a process executes an EQUIP XREQ that specifies device type only, the system essentially creates a logical device of that type out of the pool of "free storage" disk blocks allowed to the account, and then attaches the virtual channel to it. Such a logical device need not (and does not) have a specific name, since the process accesses it through the lun that was equipped to it originally. It is known as an "unsaved device", and (if a .SAF, .RAF, or .CODE) may be used as a scratch file by the executing process; for example, a sort program may equip several .SAFs for temporary work files, writing and later reading records to and from them. When a process unequips an unsaved device, that device is lost, and its blocks revert back to free storage.

To create an element of an account library (i.e., to save a logical device), a process first equips to an unsaved device of the proper type as above and then, either before or after using it for I/O as an unsaved device, executes a SAVE XREQ which specifies the lun, the account name (normally one's own), the name to be given to the device, and possibly other attributes to be assigned to that specific device.

Conversely, a saved device is destroyed by first equipping to it, executing the UNSAVE XREQ, and finally unequipping.

6.5 Device Access

A logical device may possess one of several access modes. The mode on a specific device depends on its device type and, for a saved device, the access mode set on that device by an account that had the authority to do so. In addition, a logical device also has a "current" access mode as viewed by a process which has equipped a lun to it; this depends on the above factors plus what was requested by the EQUIP, and applies to unsaved devices as well. Possible modes are:

- o Read/Write (may be equipped for write by only one process at a time)
- o Read Only (data protected from modification and destruction)
- o Append Only (read disallowed)
- o Execute Only (for .CODEs only)
- o Multiple Write (read or write; may be equipped for write by several processes simultaneously)
- o Destructive Read (read only; the .SAF is being destroyed as it is being read -- read once only)
- o No Data Access (usually a temporary mode for special purposes)

The legality of an EQUIP, and the subsequent actual access allowed over an equipped lun, depend on the device access mode and the type of equip executed; the four types of EQUIP are:

- o equip for read only
- o equip for read/write
- o equip for multiple write
- o equip for no data access

Both static and dynamic (operation-to-operation) statuses of an equipped lun may be sampled at any time by the equipping process. The following information is available:

- Device Type
- Current Access Mode
- Privilege Code (the highest privilege the process has over the device and its data, as follows:
  - Read Only
  - Read/Write
  - Modify -- the process can modify device attributes, and has complete control over the device)
- Load Point (flag)
- Saved Device (flag)
- File Writing When System Crashed (flag)
- Record Type (of last record read)
- Error Flag and Error Code
- (the current position pointer on a .RAF or .CODE is also available)

6.6 Device Types

.TERM:

.TERM is the terminal associated with an interactive (primary) process. Records of various types may be written onto a .TERM, and records (terminated by selectable input characters) may be read from a .TERM. .TERM has the following attributes associated with it:

- Lines Per Page
- Carriage Width
- Rate (standard baud rates between 110 and 19,200)
- Carriage Return Delay
- Line Feed Delay
- Form Feed Delay
- Delete Character (default is backspace = Control-H = BS = ASCII 8)
- Line Kill Character (default is line cancel = Control-X = CAN = ASCII 24)
- Echo (on, off)
- Terminating Character Group (carriage return only, or refer to table)
- User Defined Terminating Character (table -- if not CR only)
- Terminal Type (includes pointer into a public library file with more information; for use by Virtual Terminal software package, which provides terminal type independence)

.LP:

A .LP (meaning line printer) is a record (line) oriented output device. All outputs to a .LP (as well as to other record oriented output devices) are written onto a disk file and when the file is unequipped, the output is queued to be printed on some physical line printer. The creation of this "virtual line printer" on the disk allows many processes to be writing onto .LPs at the same time. Unlike a regular sequential disk file, a .LP may not be backspaced, rewound, etc. The only control operations allowed are RELEASE (which destroys all information so far written onto the .LP, so that when it is unequipped nothing will be printed), Write File Mark (a filemark is printed as a page eject), and, of course, STATUS and CLEAR.

.MT:

.MT stands for (nine-track) magnetic tape. An EQUIP to .MT causes the system to look for the presence of an interactive user logged on as the system operator. The Read Only or Read/Write parameter included with the EQUIP allows the system to inform the operator whether to make the tape read-only or writable;

a tape reel identifier (or "scratch reel" code) is also included with the EQUIP, and the system passes this information to the operator.

The user process is suspended while waiting for the EQUIP to be completed, since operator attention is required. The system operator informs the system of which tape drive he has set up in response to the request; the system checks tape labels if appropriate to verify the reel identification, and then completes the EQUIP.

A process may request the use of several .MTs; it distinguishes among them by lun.

.NULL:

.NULL is a special output device that requires no storage, and absorbs anything written to it. Any record written to .NULL is accepted and forgotten; it cannot be retrieved. (A read from .NULL always returns end-of-data.) The utility of .NULL is in program debugging and testing, where a program produces an output stream on some lun, and that output is not of interest during debugging.

.TASK:

The system performs batch processing through virtual, spooled card readers. .TASK is the view of these devices' "input hoppers" provided to processes that wish to submit a "deck" for batch processing. A process equips one of several, named .TASKs and then writes records to it in sequential fashion; each record written is analogous to a card added to the deck. When the deck is complete, the process unequips the .TASK to submit it. At unequip time, the deck is automatically queued for batch processing. Prior to the unequip, the writing process may perform the RELEASE XREQ, and of course STATUS and CLEAR.

The pseudo-file created on this logical device, between equip and unequip, should be logically equivalent to a series of lines typed at an interactive terminal during a complete interactive session; that is, the first record should be the Control Mode HELLO command, and each subsequent record should be the equivalent of a line that could have been typed at a terminal, with the session properly closed out with a log-off. (Actually, Control Mode DO-file techniques can also be used in batch, and a deck may consist of more than one "session"; see Section 4.)

<u>.CDR</u>:

.CDR is a logical device seen only by a batch job, as its standard (lun 1) and fixed (lun 201) input device. It is the "other view" of a .TASK, the running batch program's view of the virtual card reader. (Note that standard -- lun 2 -- and fixed -- lun 202 -- output devices for batch processes are .LP.)

<u>.DIR</u>:

.DIR is a pseudo-device that is used for accessing the directory of an account library. A program equips to .DIR of a specified account (normally its own) and then reads from it. (Writes give errors, since only the Monitor can change a directory.) What one reads are the directory entries for that directory. The number of entries one gets depends on the size of the buffer associated with the READ XREQ; an integral number of entries will be placed into the buffer and the last entry fetched will be remembered, so that the next read can continue from where the last left off. Thus, a program can pick up directory entries one at a time or perhaps hundreds at a time (the latter case involves tremendously lower overhead per entry fetched).

The first read of the equipped .DIR gives a "header" that describes the account.

.SAF:

A .SAF (Sequential Access File) looks like a simulated magnetic tape. A process may write records of variable length (zero to 512 Kcharacters) or write filemarks, read records or filemarks, and perform assorted operations such as space forward one record, space backward one record, rewind (set to load point), skip forward past filemark, etc. A .SAF thus has a beginning (called a "load point"), followed by any number of variable-length records and file marks, and an end (called the "end of data"). When working with a .SAF there is a pointer into it that gives the current position, much like a magnetic tape that has the read head positioned between two records somewhere on the tape. If a .SAF is rewound (set so that the position is just before the first record) and three records are read, then the current position is between the third and fourth records. A filemark is like a record in the sense that after reading one, the current position is just beyond the filemark, and after a backspace of one record (over a filemark), the position is just before the filemark.

Records on a .SAF are between zero and 2**19-1 (524,287) characters long. To each record is added 6 characters by the system to hold the length and type of the record. These length fields are not seen by user processes using the normal read and write XREQs, but they do occupy space in the .SAF. A filemark is three characters long. A .SAF is allocated on mass storage by blocks, so the file grows and shrinks by whole blocks. A block holds 4096 characters (one page), and the maximum number of blocks a .SAF may contain is the lesser of 2**20 and the size of the volume that contains the .SAF.

.SAFs are record-oriented devices, and the record types of the records written onto them are retained. The system does not interpret the type of record written on a .SAF, but it does make the type available to a program which reads the record.

.RAF:

A .RAF (Random Access File) looks to the user process like a continuous string of 2**32 characters (exactly) within which the user process can position and read and write a variable number of characters. The system implements this structure by up to two levels of index blocks pointing to the data blocks, but the user process sees only the data blocks. A data block is allocated only when data is placed in it; thus a .RAF may have "holes" which contain all zero bits (apparently) if a process reads from them, but do not use any disk space until something is written into them. A process which writes a single character at character position one hundred, and another single character at character position one hundred thousand, will cause only two data blocks to be allocated.

The normal operation of a .RAF consists of "seeking" (positioning) to some character position within the .RAF and then using the normal READ and WRITE XREQs to read or write a string of characters. When either a read or write is done, the current position is advanced over the data transferred, so that continuous reads or writes (in any combination) will progress sequentially through the .RAF.

A .RAF is not a record-oriented structure, so the type of a record written onto a .RAF is ignored (no type field is actually written); a group of characters read from a .RAF always returns a "binary" record type indicator.

The MAPIN XREQ maps in a block of a .RAF onto a page of process address space. A change to that page in memory changes the data on the .RAF. This is an extremely fast and efficient way of modifying a .RAF, and is also one way to effect communication between processes. (Note: as networks of BTI 8000 systems are implemented, separate processes on different systems within the network will not be able to communicate in this manner.)

A saved .RAF can have the access modes of read only, read/write, or multiple write -- in the latter case, many processes may reference the .RAF with update capability simultaneously.

The .RAF structure, with its addressing capability, is provided by the system as a building block for complex file structures; in particular, the .RAF addressing operation (positioning to a character address) is designed as the most general possible base mode (or "primitive operation") for storage access. It is the function of the language run-time environments, and of course the Data Base Management System, to provide more sophisticated file access techniques (relative record, indexed, etc.) "on top of" .RAF addressing.

.CODE:

A .CODE is a special form of a .RAF that can contain only 2**22 characters. The first 2**19 characters are considered to be an image of the executable code space of a program, to be run in the virtual address space of a process. The remainder of the .CODE may be used for symbolic information about the program, thus carrying debugging information with the program itself. An XREQ causes a .CODE to be run as a program; .CODEs are normally created by the loader, and normally contain only a program image and associated information.

A .CODE can be made "execute only", which allows access only for the purpose of running the program it represents. No read or write operations, nor any other operation that would allow one to see the program or its data, are allowed in this case.

A .CODE is accessed in exactly the same fashion as a .RAF (except for the smaller upper bound on .CODE addresses). Also like .RAFs, .CODEs may have their pages mapped directly into process address space for direct reference (unless, of course, the .CODE is execute-only).

.PATH:

A .PATH is a special inter-process communication device. A .PATH is saved in the account library under some name, and then is shared with whatever other accounts whose processes one wishes to communicate with. Once the .PATH is saved, only one process may equip it to read it, but more than one process may equip it for writing. The writing processes then write messages onto the .PATH (in the form of records), and the system queues these messages to be read by the (single) reading process. If too many messages are in the .PATH queue, the next writer will be suspended until some of the messages are read (a process may time itself out of such a suspension). If a reading process reads from an empty .PATH, it is suspended until a message appears in the queue (with timeout allowed in this case also).

If no messages have passed through a .PATH and one end of the .PATH has not been equipped, then, when a process at the equipped end performs some operation on the .PATH, that process is suspended until the other end of the .PATH is equipped, at which time processing continues normally. If, on the other hand, messages have been passed through the .PATH and one end of it becomes unequipped, then if a process at the other end performs some operation (other than UNEQUIP) on the .PATH, an "abnormal-end-of-data" status is returned, signifying that the communications path has been broken.

If a .PATH is given the access mode of Read/Write, then there may be only one writing process. If it is given the access mode of Multiple Write, there may be more than one writer; in this case, the users must agree on protocol (e.g., how to identify the author of a message).

In the status indications for a .PATH, the load point flag for the writer and the end-of-data flag for the reader indicate an empty .PATH. A user interrupt may be armed and enabled to cause a trap whenever a .PATH being read from becomes non-empty.

.LOCK:

A .LOCK is a device used for the coordination of cooperating processes, as a convenient and efficient alternative to mapping in a common page of a multiple-write .RAF. .LOCKs should be shared with multiple-write access. They are two-state devices: a .LOCK may be either "locked" or "unlocked".

When a process performs a READ on a .LOCK, and the device is "unlocked", the system changes its state to "locked" and allows the process to continue (nothing is returned by the READ). If, however, the state is "locked", then the system suspends execution of the reading (locking) process and puts it on a queue along with any other processes which attempted the same operation. When the .LOCK becomes unlocked, the process at the head of the queue is reawakened, with the .LOCK again locked. A suspended process which wakes up ahead of time because of a timer interrupt will be taken off the queue.

The WRITE operation changes the state of a .LOCK to "unlocked". (Note: the termination of a process "holding the lock" causes an implied WRITE on the .LOCK.)

The state of a .LOCK can be tested with a STATUS operation: "load point" indicates that the .LOCK is unlocked.

6.7 I/O Operations

The following XREQs, many of which include several parameters, operate on logical devices through a virtual channel (an equipped lun):

- EQUIP, UNEQUIP: Make/break association of the process with a logical device through a virtual channel. A device must be equipped to perform any of the following operations on it.
- STATUS: Get status of device/lun.
- CLEAR: Clear error conditions on an equipped device.
- REWIND: Rewind device to load point.
- FWSP, BKSP: Forward space/backspace one record.
- SEFF, SEFB: Space to end of file (filemark) forward/backward.
- SEOD: Space forward to end of data.
- WFM: Write filemark.
- RELEASE: Release (throw away) all data on the device.
- WRITEx (x=T,TF,B,L,C): Write a record of the specified type from a buffer of specified location and length; types are text, text with formatting, binary, label, comment.
- READ: Read a record into the specified buffer location.
- ACCESSx (x=W,R,AO,XO,MW,DR): Change access mode of device to that specified (W=Read/Write, etc.).
- POSITION: Set current position pointer in a .RAF/.CODE.
- MAPIN, MAPINR: Map in a page of a .RAF/.CODE into process memory (with or without Read-Only protection).
- RAFLAST: Get last data address in a .RAF/.CODE.
- RAFPROT: Protect a page of a .RAF/.CODE from modification.
- SAVE, UNSAVE: Install/remove a device in/from an account library.
- SHARE, UNSHARE: Grant/revoke saved device access to other account(s).
- MODIFY: Modify certain attributes of a saved device.

6.8 Saved Files

The term "saved file" refers to a logical device of type .SAF, .RAF, .CODE, .PATH, or .LOCK which has been made an element of an account library with a SAVE XREQ. A saved file remains intact when a process unequips it, and also when a user logs off the account; given proper access privileges, it can be accessed by processes running under other accounts.

A saved file identifier ("file ID") consists of a filename of from one to twelve characters optionally followed by an extension. The extension is added by following the filename with a period (".") and then up to six characters. The extensions, which are not to be confused with names of logical device types (.SAF, .RAF, etc.), are provided for the benefit of user software (including BTI-supplied language translators and utilities), to give programs an indication of the purpose or content of the saved files. For example, BTI-supplied software recognizes the following extensions, among others:

.ASSEM	assembly language source
.FTN	FORTRAN language source
.COBOL	COBOL language source
.BASIC	BASIC-X language source
.PAS	PASCAL-X language source
.RPG	RPG II language source
.LIST	a source listing of a program (produced by a compiler, etc.)
.OBJ	one or more object modules acceptable to the loader
.LIB	like .OBJ, but treated as an object module library
.CODE	loader output; executable form of a program
.TEMP	a temporary file to be destroyed soon

For example, if a programmer were working with a COBOL program named BILL, the BTI-supplied COBOL compiler and associated software would automatically recognize BILL.COBOL, BILL.OBJ, BILL.LIST and BILL.CODE. The user would not normally be concerned with any name except "BILL" itself.

Every saved file possesses the following attributes, many of which may be changed by various XREQs. Complete information about a saved file's attributes is available in the directory (.DIR) of the owning account's library; a subset is available through an equip to the saved file.

- File ID (filename and extension)
- Type (logical device type: .SAF, .RAF. .CODE, .PATH, or .LOCK)
- Creation Date
- Last Accessed Date
- Last Modified Date
- Last Backup Date
- Purge Interval: This is a number from 0 to 255 giving the number of days from the file's last access until it may be purged (removed from disk storage and written onto some archival medium). Zero specifies no purge allowed. The default is a system manager parameter, with 90 days considered reasonable. A purge interval of a week or less (1 to 7) specifies a temporary saved file which is to be removed from storage without backup after the specified interval.
- Purged Flag: When a file is purged, its directory entry remains, with this flag set. Processes attempting to equip to a purged file are notified where the file may be found.

- Access Mode
- No Backup Flag (for non-critical files)
- Recovered From Backup Flag (to warn that data might not be current)
- Shared List: This is the repository of information that allows other accounts to access this saved file in some way. Each entry in the list consists of an account ID, a "shared access code" which defines what kind of access is granted, and an optional password.

  If the account ID is that of the owning account's project public library (.dddppp), then access is granted to any account in the project. If the account ID is that of the owning account's division public library (.ddd), then access is granted to any account in the division. If the account ID is that of the system public library (.SYS), then access is granted to any account on the system.

  There are three Shared Access Codes:

  Read Only: No matter what is allowed by the file's Access Mode, only reads are allowed to the account(s) with whom the file is shared.

  Read/Write: The account(s) is(are) allowed whatever access is allowed by the file's Access Mode.

  Modify: The account(s) is(are) allowed the same privileges over the file as the owning account, including authority to modify its attributes.

When an account attempts to equip to a saved file, the shared list is searched and, if the account ID is not found in the list or the account ID is found but the password provided with the EQUIP does not match, the equipping process is given no indication that the file exists.

There is also a shared list that applies to an entire account library. Placing an entry in this list is equivalent to placing that entry in the shared list of every saved file in the account library. This is most often done when an archival account is created, where some other account (usually belonging to the same individual) has Modify privileges over the entire library of the archival account.

7. CONTROL MODE, ASSEMBLER, AND UTILITIES

7.1 Control Mode

The BTI-supplied Control Mode program is the fundamental interface between the system and all its users. It combines the functions of an interactive command language and a traditional (batch processing) job control language. The term "Control Mode" may refer either to the program or to its use, so that an interactive user conversing with the program (or a batch stream submitting command records to it) is said to be "in" Control Mode.

Control Mode is a "user" program, as opposed to being part of the Monitor; that is, it runs the same user-mode CPU instruction set that a user-written program does, and in other ways uses the same virtual machine process environment as a user-written program. It accomplishes its design functions through the same XREQ pseudo-instructions that are available to other programs (with one exception). The distinction of Control Mode is that it is known to the Monitor as that program to run in a user process environment when no other program is running; it is the "overmost" program in every process structure, as presented in Section 5. It is also given the unique capability of executing the HELLO XREQ, which logs a user on to the system; this ensures that password-checking is performed properly for legal entry to the system.

For a turnkey interactive user, Control Mode serves only as the log-on control vehicle; after this user gains access to his account, the Hello Program takes charge of the interactive interface, as discussed in Section 4. Other users, including programmers, normally spend very little time in Control Mode, simply using it to start programs by which they do their work. If desired, however, Control Mode commands can be used directly as a convenient way of exercising the system's XREQ capabilities. Control Mode program logic also provides many other services to the user.

"DO file" processing is a powerful feature of Control Mode. A DO file is a series of Control Mode commands stored as a series of text records in a file. The Control Mode DO command, given the name of the file, processes that file as if it were a program written in the Control Mode quasi-programming-language; DO files can include branches, tests, and loops, passing values between Control Mode variables and the programs their commands may invoke.

Control Mode interpretation of user commands is designed to allow flexibility and ease of learning. The general appearance of a Control Mode command is a "verb" followed by parameters; users may separate the command elements with spaces, commas, semi-colons, equal signs, or almost any other special characters that the user may feel are helpful in making the command "look right" on a listing.

If the user follows a verb with a question mark (or alternatively uses the HELP verb parameterized by the verb in question), Control Mode displays an explanation of the verb.

As long as a program name does not conflict with a Control Mode verb, the user may run a program merely by typing its name (the name of the .CODE file). Thus to run a program named "PAYROLL", the user need type only PAYROLL. (The RUN verb may be used in case of conflict).

Control Mode also provides substitutuion and concatenation operators which are applied to command strings before their interpretation; this "pre-processing" provides both interactive convenience and flexibility (when, for example, a DO file is generated programmatically).

The following summary presents only the verbs used in Control Mode commands, grouped by function:

<u>Log-in and log-off</u>:	HELLO, BYE One or two Hello Programs may be associated with an account. Programs may execute the BYE XREQ directly.
<u>Assistance</u>:	?, HELP, COMMANDS Some commands have two levels of explanation available.
<u>Modifying C.M. interaction</u>:	BRIEF, DETAIL C.M. error messages are shortened when in "brief" mode.
<u>DO files and C.M. variables</u>:	DO, IF, SKIP, TAG, EXIT... A DO file (started with the DO command) may test C.M. variables with IF; SKIP to a TAG; and EXIT before end-of-file. *, COMMENT... DO files may be annotated with COMMENT records.

ABORT, UNABORT...
In ABORT mode, certain C.M. commands are unavailable, and programs may not be started. This mode is used for DO files and batch.

SET, RESET, DISPLAY
Control Mode variables, which can be used to pass information between Control Mode and the programs it runs, may be SET and DISPLAYED; RESET clears them all.

<u>General Information</u>: DATE, TRAFFIC, LIMITS, PORT, BLOCKS, TIME
Traffic displays number of ports in use, number of active processes, and a calculated system load factor; LIMITS displays account limit information: PORT can get or set port operating parameters; BLOCKS and TIME refer to session limits for use of storage and time.

<u>Account and session parameter changes</u>: PASSWORD, BLOCKS, TIME, PORT
A "future" password may be specified, to take effect at some given date.

<u>Program execution and control</u>: RUN, implied RUN...
A program may be started by entering its name.

GO, STATE...
GO resumes execution of an interrupted program; STATE displays console information, for debugging purposes.

PERMIT, RESTRICT
Modes may be entered to trap out the execution of certain "dangerous" XREQ's. Used when running a borrowed program, or when debugging.

Account library:

CATALOG, STATUS...
Parameters allow the CATALOG command to display directory information in several ways (different sequences, more or less detail, etc.). STATUS can display attributes of a specified saved file or equipped logical device.

RENAME, MODIFY, UNSAVE...
Saved files may be renamed, and certain of their attributes may be modified; they may also be unsaved (destroyed).

SHARE, UNSHARE
Access to a saved file may be granted to other accounts, with several restrictions (including equip-time password) possible.

Virtual channels:

LUNLIST, STATUS, RESET
The EQUIP status of all 202 virtual channels can be examined with one command (LUNLIST), or each lun can be examined

	individually (STATUS). RESET un-equips all virtual channels.
Logical devices (general):	EQUIP, UNEQUIP... Programs can be written to delay the selection of logical devices to run time, when users (interactive or batch) can equip their virtual channels in Control Mode just prior to execution.
	STATUS, ACCESS, SAVE The ACCESS command invokes the ACCESS XREQ to change access code on an equipped logical device or saved file.
I/O positioning operations:	POSITION, REWIND, BKSP, FWSP, SEFB, SEFF, SEOD These commands invoke the corresponding XREQ's on an equipped logical
I/O data operations:	RELEASE, WFM, DATE, LABEL, FORMS RELEASE throws away all data on a lun; WFM writes a file mark; DATE can be used to write a record containing current date and time; LABEL and FORMS write label and forms records on luns, with the content given by parameters to the commands.

7.2 Assembler

The BTI 8000 assembler (ASSEMBLER) is an extremely fast assembly-language translator whose features and internal design resemble those of a compiler rather than a traditional assembler. For example, an assembly-language program has block structure; BLOCK and END pseudo-instructions (statements) are used to define nesting, with EXT and ENTRY statements defining locality of symbol references.

ASSEMBLER relieves the programmer of the burden of selecting address modes for operands; it provides for definition of data structures, and then automatically generates the proper address modes for referencing elements of those structures. Operand-field entries refer to the operands themselves, not their addresses.

For example, the following statements provide a formal (template) definition of a data structure to be pointed to by register 6:

```
BLOCK     BASE     R6        Begin structure definition; R6 is base reg.
ALPHA     BSSB     6         A six-bit field.
BETA      BSSB     30        A thirty-bit field.
GAMMA     BSSC     5         An array of five characters.
DELTA     BSS      4         An array of four words.
          DRCT               End of template definition.
          ORG      BLOCK     Back up and recover space allocated.
```

The next instructions establish the third DELTA word as a pointer into a table of thirteen-bit elements (TABLE):

```
TABLE     VFDB     13: element value, element value, ...
PTR1      PTR      TABLE
```

```
LD    R6     -----         (address of a particular instance of
LD    R1     PTR1                                   BLOCK)
ST    R1     DELTA(2)
```

Then, the statement:

```
LD    R0     @(DELTA(2))(R2)
```

loads register zero with the contents of the thirteen-bit TABLE element found as follows: The third (0,1,2) word of the DELTA array, within the structure pointed to by register 6 is used as a pointer (@) into the TABLE; the pointer is post-indexed by register 2 (R2) to arrive at the desired operand (pointers refer to structure elements which need not be one word long). If register 2 contained the value 3, for example, the operand would be the fourth (0,1,2,3) of the thirteen-bit fields in TABLE. ASSEMBLER automatically generates the FPVR2 address mode to handle the operand specification.

Although it is a single-pass translator, ASSEMBLER allows forward referencing by deferring code generation as appropriate. It includes a BOX statement to encourage documentation; BOX encloses comments in a "box" of asterisks on the listing. The INPUT statement allows the source of language input to switch among files, with nesting of sources allowed. ASSEMBLER produces detailed cross-reference listings, with each cross-reference indication annotated to show how the symbol was used.

7.3 Linking Loader

The BTI 8000 linking loader is named LOAD. Its input is one or more object modules -- the output of compilers (or the assembler), stored in .RAFs. Its output is an executable program image, in the form of a .CODE file. Object modules from various compilers may be combined into one program.

In most cases, the LOAD program is invoked automatically by one of the compilers, as a result of a programmer's request for a combined compilation and load. If desired, however, LOAD can be run as a distinct step, as for example when a programmer has a complex set of object modules to link together. In either case, the programmer can choose to have the resulting .CODE file saved under some name (which can be supplied by the compiler automatically) or left as an unsaved device which is used for a single (immediate) program execution and then released.

All BTI-supplied language compilers (using this term to include ASSEMBLER) produce object modules in a standard format. More than one module can be stored in a .RAF, so that "libraries" of object modules can be collected into a .RAF for the convenience of the loader; a .RAF with the filename extension .OBJ (the output of a compiler) is internally indistinguishable from a .RAF with the filename extension .LIB (which indicates a collection of "library routines"). The loader collects all object modules required for the construction of a program from as many .RAFs as necessary. Standard BTI-supplied object module library .RAFs in the .!!! system public library account are accessed automatically; there are object module library .RAFs supplied for each language, plus a default library which is always searched if any unresolved symbols remain.

If necessary, the loader searches through library .RAFs repetitively (until all references are resolved), in case a new module references a module stored in an earlier position within the library .RAF.

.CODE files not only provide for the 128-page virtual address space program image, but also can include up to several hundred pages for storage of symbolic information and overlay segments.

The standard object module format includes symbolic information about the object code for ultimate use by debugger programs. The loader incorporates this symbolic information into the high pages of the .CODE file, so that an interactive debugger can refer to variables and other parts of a program by the same names used in the source program. (When a program is executing, the .CODE file is always equipped, on lun 200.)

The loader collects object module "sections" of like types into program sections; it then places these sections, in both address space segments and overlay segments, into the .CODE file in a specific arrangement. The type of a section is determined by four characteristics:

PURE/IMPURE:	Pure sections are placed into write-protected pages.
PAGE ORIENTED/WORD ORIENTED:	This refers to requirements for boundary alignment.
REUSABLE/ADD-ON:	A reusable program section consists of enough space to contain the largest of its contributing module sections; an add-on program section requires space enough to hold all contributing module sections in sequence.

GLOBAL/LOCAL: A local section is one which has meaning only within a given overlay segment.

For example, ordinary program instructions are placed into a section which is PURE, WORD-ORIENTED, ADD-ON, and LOCAL; a FORTRAN COMMON area is placed into a section which is IMPURE, WORD-ORIENTED, REUSABLE, and GLOBAL.

The loader is responsible for creating the final overlay schema, which it places in page zero of the .CODE file. The schema is a tree-structured set of information which references the overlay segments placed in the high pages of the .CODE file.

A special feature of the loader is the linking of "shared run-time systems". Rather than link-loading language run-time environments separately for each program that requires them, the loader relocates sharable code such that the pure sections of each language's run-time system can be shared by all processes that require that environment.

Finally, the loader contains a feature known as "the calculator". Due to the richness of the CPU address mode set, evaluation of unresolved externals in link-loading may require calculations much more complicated than the simple substitution of a relocated address. The loader's "calculator" uses reverse Polish notation, variable bit precision, and stack orientation to perform the integer arithmetic necessary.

An auxiliary utility program, named OBJEDIT, is provided to manipulate object library .RAFs. This program allows users to interactively examine the contents of an object module library .RAF; perform deletions, insertions, and replacements of object modules within such a .RAF; and combine the contents of several object module library .RAFs into one (or conversely, break apart one such .RAF and distribute its object modules to other .RAFs).

7.4 Editor

The BTI 8000 EDIT program is designed to accommodate a wide spectrum of users, tasks, and interactive environments by selecting one of a series of operational modes. Considerations for selection include:

User task:

(a) programmer work: program development, generation of test data, documentation
(b) "letter writer" work: secretarial; text usually limited to a few pages
(c) large documents

Terminal type:

(a) simplest interactive terminals (CRT or hard-copy)
(b) CRT's with cursor control (only)
(c) semi-intelligent terminals (hard-copy or CRT)

Display speed:

(a) low speed connection (e.g., 30 cps)
(b) high baud rate (usually hard-wired)

User sophistication:

(a) novice user (simple command subset)
(b) occasional user
(c) extensive user (uses sophisticated features)

Among the editor's features are the provision of extensive user-assistance capabilities; command synonyms (if the user is more comfortable with alternate command words, he can declare his own synonyms); user-defined macro commands (a program-like sequence of editor commands, to be used repetitively, which can be invoked with a single command -- the user defines and creates his own macros); and a choice of English-like (verb-modifiers) commands ("VERB" mode) or minimum-keystroke commands ("TERSE" mode). In TERSE mode, the editor informs the Monitor of those (alphabetic) characters to accept as "input line terminators", so the system need not incur the overhead of calling in the program to process every individual input character. (Hence in TERSE mode, the parameters, if any, precede the one-or-two-letter command.)

The editor can operate on several files at once; each is normally a .RAF (but can be a .SAF), organized into "text pages", which are collections of lines (not limited to one printing page). There is also a special "table of contents" at the front of the file which can be searched to locate a text page by number or by a content search based on the first text line of the text page. (The table of contents is written using record forms which are not normally seen, except on request, by most user programs.) Lines can be moved across text-page boundaries, and text pages can be rearranged within files or even transferred among files.

In "display mode", as opposed to "line mode", the CRT display (or hard-copy print-out) is always formatted in the way a final print-out would appear, as far as possible; the results of an editing operation, in most cases, are displayed immediately. Display mode operates on text using the concepts of a "window" into the text (normally set at the dimensions of the CRT screen), plus a "cursor" which identifies the "current line" within the current window. Intra-line editing can be done by inserting, deleting, or rewriting text on the current line, in place, with changes reflected immediately.

For rearranging or inserting lines of text, the user can "attach" a series of lines to the cursor; as he moves the cursor within the text, these attached lines follow it, visually as well as logically; the user can actually see what the text would look like with the "attached" lines appearing in different positions, since the display is updated with every cursor or window movement. A "release" command deposits the attached lines into the text at the current cursor position.

Lines (which are logical lines not limited in length) can be addressed by line number within page; by context (cursor position plus or minus "n" lines); or by content (the line or lines containing the characters "xxx..."). Line ranges are identified either by starting and ending line addresses (as above), or by starting address plus line count. String searches can be "delimited"; the editor can distinguish between, for example, "the" as a separate word, and "the" as a string that may appear as a piece of a word. Intra-line modifications, as well as whole line deletions, can be "undone".

The editor can also perform several types of justification and centering operations (left, right, both, center), and can break apart a "paragraph" into its component sentences, creating a separate edit-line for each sentence; the user can then rearrange, add, or delete sentences, and finally repack (and justify) the paragraph.

7.5 Sort and Merge

The BTI-supplied SORT utility program can be run either as a standalone program or as a subroutine whose object module is linked into a user program by the loader. SORT accepts input from as many as twelve logical devices -- these are normally record-oriented devices, but may be .RAFs. There is no limit to the number of input records; maximum input record size is 99,999 characters. The sort algorithm used is a sequence-preserving binary tree sort.

SORT instructions ("directives") may be entered from the terminal or stored on a file. If a user types only SORT in Control Mode, the program prompts the user for directives. If the user instead enters

```
SORT I=i, L=l
```

then SORT expects to find its directives stored on the device specified by the "I" parameter ("i" represents a lun or a filename), and will list its statistics and error messages on the device specified by the "L" parameter. If SORT is called as a linked subroutine instead of as a standalone program, the I and L parameters are passed with the subroutine call.

The directives, whether entered from the terminal or read from a file, are as follows:

BLK= number of records per block for input devices; zero means variable-length records, unblocked (this is the default if the BLK directive is omitted).

LEN= record length in characters (if fixed-length records); default is "80", if the LEN directive is omitted.

INP= lun for input source. There may be up to 32 INP directives, specifying up to 32 sources of input. If the blocking factor for a particular device is other than that set by the BLK directive (or its default), the specification is made by adding a modifier: INP=i/BLK=b.

OUT= lun for output. The blocking factor is the same as that used for input devices, uless specified otherwise: OUT=o/BLK=b.

SORT may be given two OUT directives, the second specifying the lun on which to write records whose keys are duplicates of those of previously encountered records (records with such duplicate keys are called "dupe records"), if so specified by the DUP directive:

DUP=PICK	Causes dupe records to be written to the second OUT lun.
DUP=DROP	Causes dupe records to be "forgotten".
DUP=OK	Causes dupe records to be written on the first OUT lun, behind their primaries. This is the default if the DUP directive is omitted.

(In all three cases, adding the modifier /LIST causes the first 80 characters of dupe records to be written to the listing device.)

The FLD ("field") directive names a field within the record, identified by starting character position and length in characters; this directive may be used up to 15 times, naming up to 15 fields: FLD(fieldname)=start,length.

The KEY directive identifies the fields (named by FLD) that are to be used as sort keys. Up to 15 KEY directives may be used; the order of their appearance specifies their relative importance (major to minor): KEY=fieldname. The following modifiers may be used after "fieldname":

/ASC	ASCII character interpretation (default, if no modifier).
/ASL	As above, but lower-case ASCII characters are treated as if they were upper-case characters.
/SBN	(Signed binary) The high bit of the first character of the field is treated as a sign bit, and the remainder of the bits in the field are taken as an unsigned binary number.
/UBN	(Unsigned binary) Straight binary comparison of all bits in the field.
/-ASC, /-ASL, /-SBN, /-UBN	As above, but sort sequence is descending instead of ascending.

Finally, the GO directive ends the list of directives and begins the sort:

```
GO
```

The merging phase of SORT equips scratch .SAFs as necessary, entering a new merge phase automatically after using 32 merge files. It uses a tournament merge that dynamically balances its binary tree.

SORT writes the following information to its listing device:

All directives
Number of records read from each input device
Total number of records sorted
Total number of records output (may be less because of DUP)
List of dupe records (first 80 characters of each)

The MERGE program is similar to SORT, using exactly the same parameters and directives, and treating duplicate keys in the same manner. Each of its input files, of course, must already be in sorted sequence.

7.6 COPY

The COPY program is used for general-purpose data movement in the BTI 8000 I/O system. It copies files or parts of files, including data extracted from .SAFs between filemarks. COPY accommodates all logical device types, and, given specific definition of a record, can move data from byte-oriented devices (.RAF, .CODE) to record-oriented devices (.SAF, etc.).

A special capability of the COPY program is format conversion from and to "foreign" magnetic tapes, including code conversion (e.g., EBCDIC/ASCII), record-type conversion, blocking and deblocking, etc. COPY is designed to convert from and to most of the commonly used magnetic tape formats.

COPY can be used as an interactive program from Control Mode, or can be run as an underprogram of a user-written calling program.

8. STANDARD PROGRAMMING LANGUAGES

8.1 Introduction

The BTI 8000 is designed to increase programmer productivity by providing an extremely flexible program development environment. Any number of program development tasks in any combination of languages may proceed concurrently, without conflict. Further, the BTI 8000 offers a wide selection of standard languages to allow applications to be implemented in a language which suits the problem at hand.

The BTI 8000 supports six programming languages: COBOL, FORTRAN, PASCAL-X, BASIC-X, RPG II, and ASSEMBLER. All language implementations have the following philosophical concepts in common:

- o Program development may occur in an interactive mode. The program may be written, compiled, and linked from the terminal. Test files can be defined, built, and dumped from the terminal; programs can be tested with the aid of an interactive, symbolic level debugging facility at the terminal.

  In this environment, any number of program development tasks (text editing, compilation, program test, file build, file dump, etc.) can occur concurrently. It is estimated that interactive program development tools can increase programmer productivity by as much as 50% over traditional methods.

- All languages support terminal devices as standard I/O devices. A terminal (CRT or teleprinter) can be accessed under control of the BTI Monitor without special telecommunications software. An application program need merely execute a high level language "READ" or "WRITE" command against a file which has been EQUIPed to a .TERM device type.

- All language run-time systems, with the exception of BASIC-X, are based on a common object program format. As a result, subroutines written in a language may be linked into an object program written in another language.

- All language implementations, except BASIC-X are built upon the same data management primitives. All files can be read by all languages except BASIC-X; all Input/Output aspects of the virtual machine environment are common to all languages.

8.2 PASCAL-X

PASCAL is a high-level block-structured procedural language which has gained considerable acceptance in contemporary computing. It is particularly effective in environments which require the coordinated development of large, multi-implementor software projects, including operating systems and compilers. PASCAL evolved out of an effort by Working Group 2.1 of IFIP to define a successor language to ALGOL 60; in 1965, this group produced the language ALGOL W, the direct predecessor to PASCAL. PASCAL was specified by Niklaus Wirth at the Institut für Informatik, Zurich, in 1968; the first compiler became operational in 1970 and was published in 1971. A second version of the specification was released in 1973, and is now generally referred to as "standard PASCAL".

PASCAL was the first major programming language to be developed subsequent to the formalization of the concept of structured programming. "Structured programming" is a formal approach to program design which attempts to increase the clarity and correctness of programmed solutions through the use of top-down implementation techniques. PASCAL is block-structured and procedure-oriented, and provides the programmer with a comprehensive set of single-entry/single-exit control structures. The use of these mechanisms greatly simplifies the implementation of structured programs.

PASCAL is a sparse language with relatively few basic constructs. The power of the language stems from the manner in which the basic constructs can be combined. In particular, both algorithms and data structures can be specified hierarchically. The clarity inherent in the sparseness of PASCAL is a central design feature which provides the following benefits:

- The simplicity and conciseness of PASCAL constructs simplifies the design and implementation of compilers to support the language. Further, PASCAL compilers tend to generate extremely efficient machine code, since an underlying design objective of the language limited the allowable set of constructs to those which could be efficiently supported on existing computer hardware.

- Well-written PASCAL programs are easy to read and understand. The language allows unlimited-length data and procedure names, and exploitation of this feature allows programmers to write programs which are virtually self-documenting. Further, the very nature of the language syntax acts to limit the number of approaches which the programmer may employ to achieve a particular processing requirement -- there are fewer "correct" ways to code the program, and the clarity of the solution is improved.

- The block structure of PASCAL encourages top-down development techniques to be applied in a natural and simple manner. This type of language organization allows the experienced programmer to conceptualize a top-down solution in the actual language to be used to program that solution.

- Support of single-entry/single-exit control structures enhances the ability of a programmer to verify the correctness of a programmed solution. Thoughtful application of structured programming techniques greatly simplifies program and procedure assurance testing.

In order to discourage future compiler implementors from adding extraneous features which might compromise the above benefits, Dr. Wirth originated the idea of strict standardization of the language known as "standard PASCAL". By convention, therefore, implementations which in any way deviate from the original language specifications may not be referred to as "PASCAL" -- even if these deviations are deemed necessary to achieve mandatory implementation requirements.

There are three commonly recognized weaknesses in standard PASCAL:

- o No support of a character-string data type.
- o No provision for error-handling procedures.
- o No support of random access files of any kind.

The BTI 8000 implementation of PASCAL extends standard PASCAL to correct these deficiencies. By convention, therefore, this language cannot be referred to as "PASCAL". BTI 8000 PASCAL-X ("X" indicating "extended") does, however, include all the features of standard PASCAL; formally speaking, it is a valid superset of standard PASCAL -- one can write any standard PASCAL program in PASCAL-X.

PASCAL-X adds the following features to standard PASCAL to correct the above weaknesses:

- o Full support for STRING data type.
- o Conditional error recovery through the "ON error-condition DO" construct. In keeping with the block-structured nature of PASCAL, PASCAL-X includes a "TRY...ELSE" construct for structured error handling.
- o Full access to all aspects of the BTI 8000 I/O and file system.

Additional extensions to standard PASCAL include:

- o Run-time adjustable arrays.
- o Compile-time expression evaluation for "CONST" declarations.
- o Spawning and management of concurrent processes, and generation and control of underprograms (see Section 5).
- o Inter-process communication capability (.PATHs and shared .RAFs).

PASCAL-X also has full access to the entire BTI 8000 virtual machine. Efficiency-critical portions of a solution can be coded in assembly language, but the assembly-language routines are treated as PASCAL procedures; the block structure of the language is maintained, and machine-dependent code is isolated for ease of maintenance and conversion. The capability of using the host machine to its full capacity provides two significant benefits to PASCAL-X:

- o It becomes an ideal vehicle for systems programming (all BTI 8000 systems software is written in PASCAL-X); and
- o it allows PASCAL to be a feasible production language, providing excellent structured programming and machine efficiency capabilities to the data processing community at large.

8.3 COBOL

COBOL (COmmon Business Oriented Language) is a high level language designed for use in business data processing environments. The BTI 8000 implementation conforms to the American National Standards Institute (ANSI) specifications X3.23-1974 at the high-intermediate level.

The ANSI standard specifies the level characteristics of the language through the detailed definition of 12 language modules. BTI 8000 COBOL conforms to the standard at the following level:

Nucleus	Nucleus Level 2 provides full facilities for qualification, punctuation, characters, data-name formation, connectives and figurative constants. Within the Procedure Division, the Nucleus Level 2 provides full capabilities for the ACCEPT, ALTER, DIVIDE, DISPLAY, IF, INSPECT, MOVE, MULTIPLY, PERFORM and SUBTRACT statements. In addition to the standard, REMARK and NOTE have been retained from previous COBOL implementations.
Table Handling	Table Handling Level 2 provides a capability for accessing items in up to three-dimensional variable length tables. This level also provides the additional facilities for specifying

ascending or descending keys and permits searching a dimension of a table for an item satisfying a specified condition.

Sequential I-O

Sequential I-O Level 2 provides full facilities for the FILE-CONTROL, I-O-CONTROL, and FD entries as specified in the formats of this module. Within the Procedure Division, provides full capabilities for the CLOSE, OPEN, READ, WRITE, USE, and REWRITE statements as specified in the formats of this module. Additional features available in Level 2 include: OPTIONAL files, the RESERVE clause, SAME RECORD AREA, MULTIPLE FILE tapes, REVERSED, and EXTEND.

Relative I-O

Relative I-O Level 2 provides full facilities for the FILE-CONTROL, I-O-CONTROL, and FD entries as specified in the formats of this module. Within the Procedure Division, this level supports CLOSE, DELETE, OPEN, READ, REWRITE, START, USE, and WRITE statements as specified in the formats of this module. Additional Level 2 features include: the RESERVE clause, DYNAMIC accessing, SAME RECORD AREA, READ NEXT, and START.

Indexed I-O

Indexed I-O Level 2 provides facilities for the FILE-CONTROL, I-O-CONTROL, and FD entries as specified in the formats for this module. Within the Procedure Division, this level supports CLOSE, DELETE, OPEN, READ, REWRITE, START, USE, and WRITE statements specified for this module. Additional Level 2 features include the RESERVE clause, DYNAMIC accessing, ALTERNATE KEYS, SAME RECORD AREA, READ NEXT, and the START statement.

Sort-Merge	Sort-Merge Level 2 provides the facility for sorting one or more files, combining two or more files, one or more times within a given execution of a COBOL program.
Report-Writer	The Report Writer module provides the facility for producing reports by specifying the physical appearance of the report rather than the detailed procedures required to produce the report. The initial release of COBOL will contain a NULL implementation of the Report-Writer. A full implementation of the Report-Writer is planned for late 1979.
Library	Library Level 2 provides the facility for copying text from a source library into the source program. Level 2 Library also supports the replacement of a given literal, identifier, word, or group of words in the library text with alternate text, during the copy process. Level 2 Library supports the availability of more than one COBOL library at compile time.
Segmentation	Segmentation is not supported under the initial release of BTI 8000 COBOL.

Debug	In place of the standard DEBUG facility, BTI 8000 COBOL provides symbolic interactive debug support on all compiled COBOL programs.
Interprogam Communications	The initial release will not support COBOL-standard Inter-Program Communication constructs. An analogous capability exists as a standard feature of the BTI 8000 Monitor, which provides the additional capability to communicate between concurrently executing programs written in any language.
Communications	The Communications Module is not implemented in BTI 8000 COBOL. The standard operating system provides comprehensive support of interactive terminal devices as a standard feature of the file system; implementation of the Communications Module would compromise the device-independence features of the Monitor.

BTI 8000 COBOL includes the following enhancements to the standard:

- o Support of the discontinued 1968 COBOL constructs NOTE, EXAMINE, and REMARK.
- o DBMS-X Data Base Management System verbs are fully supported by the COBOL compiler -- no precompiler is necessary, nor is it necessary to employ the CALL verb for data base access.
- o Fully compatible data types between COBOL and FORTRAN
- o Symbolic interactive debugging facility

8.4 FORTRAN

BTI 8000 FORTRAN is a full implementation of FORTRAN-77 (ANSI standard FORTRAN X3.9-78) for use on BTI 8000 computer systems. The conpiler also accepts programs written in complience with ANSI FORTRAN 66 and commonly used extensions.

BTI 8000 FORTRAN is designed for programming convenience, allowing the programmer to concentrate on the algorithm instead of its implementation. Unlimited statement length and support of upper/lower case symbolic data names are but two of the features which support this design goal. The compiler has a three-level error and warning system for diagnostics which allows compilation to continued in spite of minor errors. Debugging statements identified by "D" in column 1 can be included in the compilation or interpreted as remarks, depending upon a selected compiler option. Finally, a fully interactive debugging facility aids in program test and verification.

BTI 8000 FORTRAN places the full power of the operating system in the hands of the programmer through the use of extended I/O facilities. Files may be created, attached, interrogated, and destroyed under program control using the OPEN, CLOSE, and INQUIRE statements. Data transfer to and from files may be formatted, unformatted or list-directed. Files may be direct access or sequential, and may contain variable length records. In addition, data may be transferred to and from character strings by using statements which are similar to regular I/O statements.

The data types and data manipulation handling capabilities of BTI 8000 FORTRAN are extremely versatile. Character strings may be concatenated and assigned to variables. Strings may be compared with other strings. Substrings may be extracted with a convenient subscript-like notation. Numeric data types (Real, Integer, and Complex) have over 15 digits of significance; Double Precision Real supports 34 digits of significance. Variables of any type may be subscripted, and an array may have up to seven dimensions with no restructions on upper and lower bounds.

BTI 8000 FORTRAN extends the FORTRAN-77 standard with the following features:

- Debugging statements indicated by "D" in column 1 included in compile by compiler option
- Array subscript and computed GO TO expressions of real, double precision or integer type.
- Interactive debugger support
- Symbolic names (1 to 8 characters in length) in upper or lower case or combination
- Subexpression optimization
- Variable length record I/O support

8.5 BASIC-X

BASIC (Beginners All-purpose Symbolic Instruction Code) is a conversational problem-solving language which has met with considerable success in the computer timesharing environment. It is characterized by its interactive, "human-engineered" user interface. BASIC provides editing facilities to support line-by-line program syntax checks, line insertion, line deletion, line resequencing and text modification from the keyboard; the distinguishing feature of BASIC is that it can immediately execute a newly entered or modified program, without the delay of a compilation phase. These features provide rapid program development for the novice as well as the experienced programmer.

BASIC-X is BTI's implementation of the BASIC language. BASIC-X is an enhanced version of BASIC ("X" indicating "extended") which serves the total application needs of over 700 installed BTI Model 3000, 4000, and 5000 Timeshared Computer Systems.

BTI 8000 BASIC-X is a totally compatible implementation of BASIC-X. The primary design objective, and in fact the reason for the development of BTI 8000 BASIC-X, is to provide BTI customers with a conversion-free growth path from the BTI 3000/4000/5000 systems to the BTI 8000. The existence of BTI 8000 BASIC-X protects the program investment of existing BTI 3000/4000/5000 owners, while allowing these customers to substantially increase their total data processing capacity.

Source-level compatibility among the various BTI product families allows multi-division organizations to use a centrally located BTI 8000 system as a program development and maintenance resource, while installing smaller BTI computers in remote locations for use as independent processing nodes. Remote locations can achieve the benefits of local computer control while allowing system support tasks to be carried out by a centrally located computation support staff.

Organizations which are geographically dispersed can implement distributed processing networks built entirely around BTI systems; distributed processing allows the organization to disperse the functional responsibility for data processing to locations where local computing power is useful, while maintaining control of the overall direction of the data processing function at a central location. Local management can assume responsibility for operation of the system, while relying on remote technical specialists to provide programming and system support.

For the most part, BASIC language implementations are interpretive in nature. At program execution time, a language interpreter accepts source program statements a line at a time, interprets the execution implications of the source statement, and performs the statement by executing appropriate machine code in the interpreter program. The interpretation of the source statement occurs each time the source line is encountered. BASIC-X on the BTI 3000/4000/5000 is implemented with an interpreter.

A second major approach to language processing calls for the use of a language compiler. Program source, which must be complete, is accepted and processed by a program (the compiler) which translates the complete source program into an equivalent "object" program which can be loaded into computer memory and executed. Both compilers and interpreters are language translators; the differences between the two include the fact that the translation process occurs once for a compiled program, with the resultant machine object code available for storage and later use without further translation. By contrast, the translation process inherent in an interpretive language occurs each time the program is run. All languages on the BTI 8000 other than BASIC-X are implemented using compilers. Obviously, compiled programs execute with far less system overhead than interpreted programs.

BTI 8000 BASIC-X is implemented using a technique which has characteristics of both a compiler and an interpreter. As source statements are presented to BASIC-X, during either keyboard entry or retrieval from a source library, each is translated on a line-by-line basis, as with an interpreter. Once the execution requirements of the line have been determined, however, the incremental compiler generates machine object code which is stored for later execution. When the program is run, the machine object code is executed in roughly the same manner that a compiled program executes (the interpreter-like features result in certain efficiency trade-offs). Further, the quasi-compiled object code can be stored and retrieved for later use, without further translation.

As implemented on the BTI Model 3000/4000/5000, BASIC-X provides a fully interactive program development cycle. The incremental compiler implementation of BASIC-X on the BTI 8000 maintains this same user environment while approaching the execution-time efficiency of a compiler.

BASIC-X includes the following extended language facilities:

- String arithmetic with 252 decimal digit precision
- Extensive string and substring manipulation facilities
- Complex error handling under control of the user program
- File creation/deletion within user program

- Up to 64 files open concurrently in each program
- Extensive file sharing features in support of update and inquiry activities
- Complete set of matrix operators
- COMmon file declaration between CHAINed programs eliminates redundant file linking
- PRINT USING to a string variable as well as to an I/O device

8.6 RPG II

RPG (Report Program Generator) II is a highly flexible, machine independent, problem-solving language. It is most frequently used in business and commercial computing environments, although the language is powerful enough to fill most application needs. RPG and RPG II are versions of a standard language available on many machines. As a result, programs written for another manufacturer's RPG can be submitted to the BTI 8000 RPG II compiler with little or no conversion re-coding.

BTI 8000 RPG II offers many of the programming capabilities of machine-oriented programming languages but, in most cases, is easier and quicker to code. The primary advantage of RPG II is the speed with which file-oriented application programs can be written.

Some of the significant advantages of BTI 8000 RPG II include:

- o The full program development cycle is supported from terminal devices. Programs can be written with the aid of the BTI Text Editor, compiled, and tested from any terminal on the system. Program Development can proceed concurrently with production processing, without degradation of the production environment.

- o Application programs written in RPG II can use terminal devices as input/output devices without additional control software. The programmer can access terminal devices through EXCPT (Output) and DSPLY (Input/Output) calculation operators. It is <u>not</u> necessary to interface the program to a special run-time control program to gain access to terminals.

The programmer may also assign standard input and/or output files to terminal devices. This approach allows the terminal to be accessed as part of the standard RPG II processing cycle.

- RPG II operates in a completely secure, multi-user environment. Program development, production processing and terminal based application programs can proceed concurrently, without special system scheduling considerations.

BTI 8000 RPG II provides an excellent vehicle to allow current batch-oriented users to upgrade to terminal-oriented data processing. Existing RPG batch application programs can be moved to the BTI 8000 with a minimum of conversion effort, and experienced RPG programmers can be trained to exploit the terminal capabilities of the system as part of the general introduction to the system.

9. DATA BASE MANAGEMENT SYSTEM

9.1 Introduction to Data Base Management

Recent advances in information science have resulted in the development of advanced techniques to aid in the management of data with the aid of computer systems. One of the most powerful tools to evolve out of this technology is the concept of data base management. The data base approach to information storage and retrieval views data as an organizational asset which can be controlled and managed with the aid of high-level system software.

A Data Base Management System (DBMS) is a library of programs, data manipulation subroutines and high-level language interfaces which allow large volumes of inter-related information to be stored on large capacity disk units. This mass of inter-related information is maintained as an entity which exists somewhat apart from actual programs and systems on the computer. The DBMS performs data management functions which control the placement of data into the physical storage domain and provide mechanisms for mapping high-level requests for data into physical storage operations. In this manner, DBMS software eliminates the need for application programs to interface directly with physical storage devices and file management systems.

DBMS-X FEATURE SUMMARY:

- o SECURE DATA STORAGE
- o PROTECTED DATA BASE INTEGRITY
  - CONSISTENT TRANSACTION PROCESSING
  - AUTOMATIC SOFT FAILURE RECOVERY
  - HARD FAILURE RECOVERY
  - INTEGRITY PROTECTION IN PROGRAM TEST MODE
- o LARGE CAPACITY DATA BASE STORAGE
- o AUTOMATIC PHYSICAL ADMINISTRATION
- o INTERACTIVE LOGICAL ADMINISTRATION

9.2 DBMS-X

DMBS-X is a general purpose data base management system supplied and supported by BTI for use on the BTI 8000 computer system. It provides for the definition, creation, maintenance, restructuring and backup of network data bases through the use of standard system functions and high level language interfaces. DBMS-X is designed to assure the integrity of information stored under its control in a multi-lingual, multi-user on-line environment. Application programs which use DBMS-X may be written in COBOL, FORTRAN, or PASCAL-X.

DBMS-X complies with and exceeds the 1973 CODASYL Programming Language Committee specifications for data base management systems. BTI has implemented significant extensions to the standard which provide the following features:

- Privacy and security controls implemented at the data item (field) level through the use of access control lists. Specific system users as identified by their account ID's are granted access to particular portions of the data base by the Data Base Administrator.

- Logical integrity protection provided through the BEGIN-TRANSACTION/COMMIT-TRANSACTION language construct. The systm assures that all operations specified between the BEGIN-TRANSACTION and COMMIT-TRANSACTION designators in an application program have been successfully completed before releasing the data base for modification by another user and/or transaction. Update sequences which fail or are aborted prematurely due to system failure, logical error, or operator intervention are completely backed out of the data base before the next user or transaction can gain control.

- A series of meta-commands in the Data Manipulation Language allows programmers to perform operations which involve groups of data sets. These commands dramatically simplify the programming tasks required to implement inquiry and reporting functions against the data base.

These enhancements to the specifications provide an implementation which is functionally superior to the CODASYL recommendations, while supplying easy conversion for software written in compliance with the standard. Some of the major features of DBMS-X are discussed below:

Secure data storage:

The benefits to be gained by providing qualified users with on-line access to sensitive data must be weighed against the risk of disclosure of that data to unqualified users. DBMS-X provides a multi-level security system which allows for the implementation of extremely tight information access controls.

At the highest level, it is impossible for an individual to gain access to any information unless he is logged on to the system as a user. As part of the log-on procedure, the user must supply the operating system with proper account identification and password information.

The log-on procedure identifies the individual to the system and to the data base management system. As part of the data base definition process, the Data Base Administrator specifies a list of users who are allowed to access the data base. The DBA also specifies the portions of the data base which can be read or changed by each valid user. In this manner, the ability to access individual data fields is controlled by the Data Base Administrator. In the absense of permission to access a data base, a system user is prohibited from any activity against the data base.

Protected data base integrity:

As large volumes of information are consolidated into an integrated, multiple-application data base, it becomes increasingly more important to assure the integrity of the data. Data base technology allows many application programs to access and update the base concurrently, and must provide tools and system features to protect the base from compromise. A major design objective of DBMS-X is the creation and maintenance of data bases which are "crash-proof" -- and to maintain consistent, coherent data bases in spite of system crashes, run-time data base control program crashes and application program crashes.

- Consistent Transaction Processing

  Almost no data base update procedure generates consistent results during all instantaneous phases of the update. Consider a financial application which debits and credits accounts -- the update procedure consists of debiting one account and crediting another for the same amount of money. At the point where the debit operation has occurred and the credit has not occurred, the contents of the data base are inconsistent. An interruption in the update at this point would leave the data base out of balance (inconsistent), and some form of remedial action would be necessary. The same type of inconsistent result would develop if a concurrently executing program deleted the credit account or changed its identifier before the debit/credit transaction was completed.

DBMS-X gives the application programmer a tool to aid in the protection of logical data base integrity in the implementation of the data base "transaction". The application programmer defines the beginning and end of an update sequence which takes the data base from one consistent state to another consistent state with BEGIN-TRANSACTION and COMMIT-TRANSACTION statements in programs which change the data base. DBMS-X temporarily blocks all other programs from accessing those resources which are required to complete a transaction, while assuring that the transaction either processes to completion, or does not process at all. It specifically prevents the system from processing part of a logical transaction, and thus protects the logical integrity of the data base. Upon completion or abortion of the transaction, control over the locked portions of the data base returns to the system.

The following sequence of events is a functional model of the transaction update process. The actual internal procedures are beyond the scope of this document.

STEP I	The application program specifies BEGIN-TRANSACTION and a list of the resources required for update.
STEP II	The required resources are locked and the Update Table Transaction Status Code is set to IN PROCESS (Figure I).
STEP III	Data base pages are changed as the update progresses, and the updated pages are stored outside the data base. Update Table entries are made which identify the locations of paired "existing" and "updated" pages. DBMS-X continues to build "shadow pages" of updated information until the application program signals the end of the logical update sequence with COMMIT-TRANSACTION (Figure III).
STEP IV	At COMMIT-TRANSACTION, the Transaction Status Code is set to CLOSE (Figure III).
STEP V	The system links the shadow pages into the data base in place of the existing pages, releasing the existing pages in the process (Figure IV).
STEP VI	The Update Table is cleared, the Transaction Status Code set to COMPLETE, and all locked resources released (Figure V).

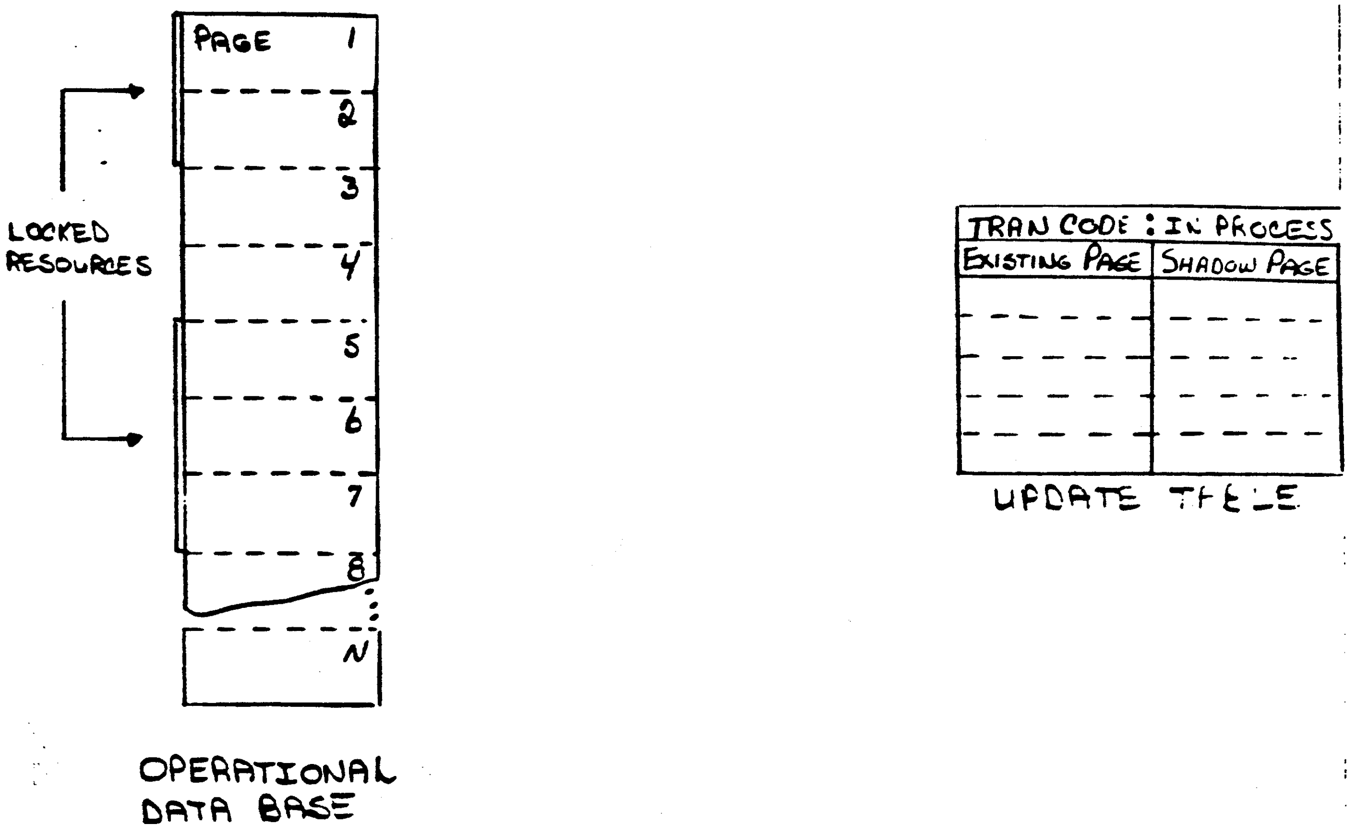


FIGURE I -- RESOURCES LOCKED, TRANSACTION <u>IN PROCESS</u>

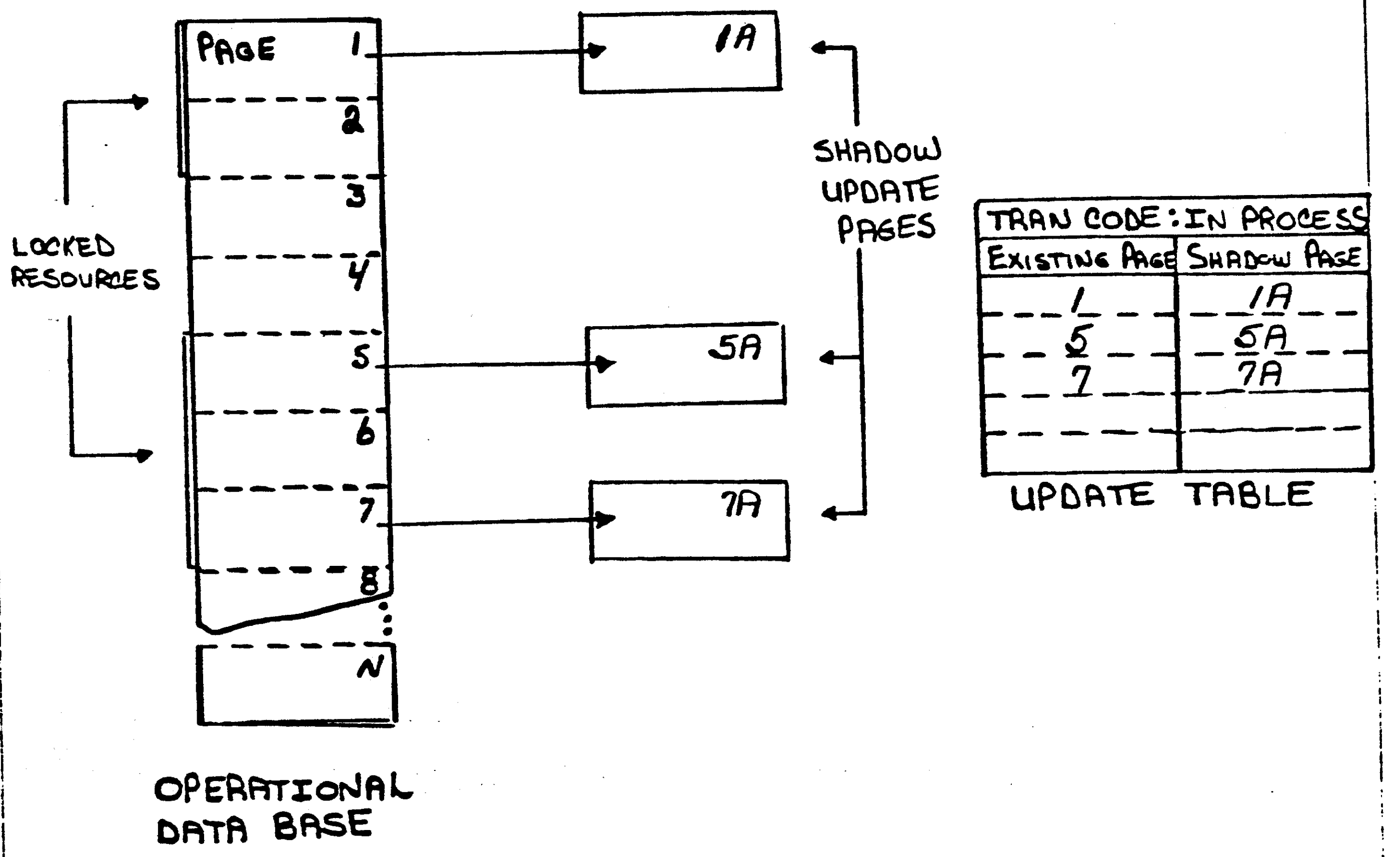
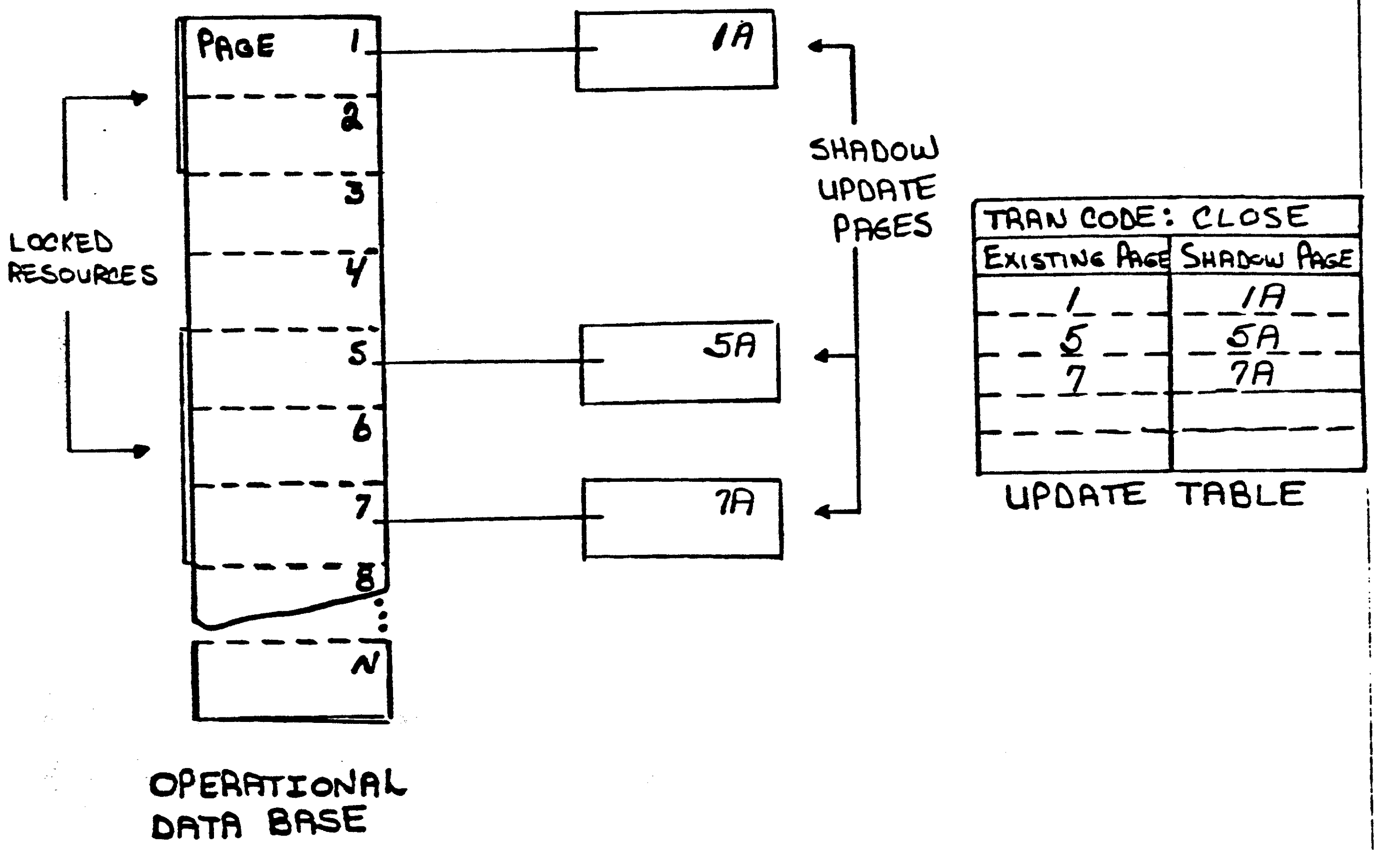
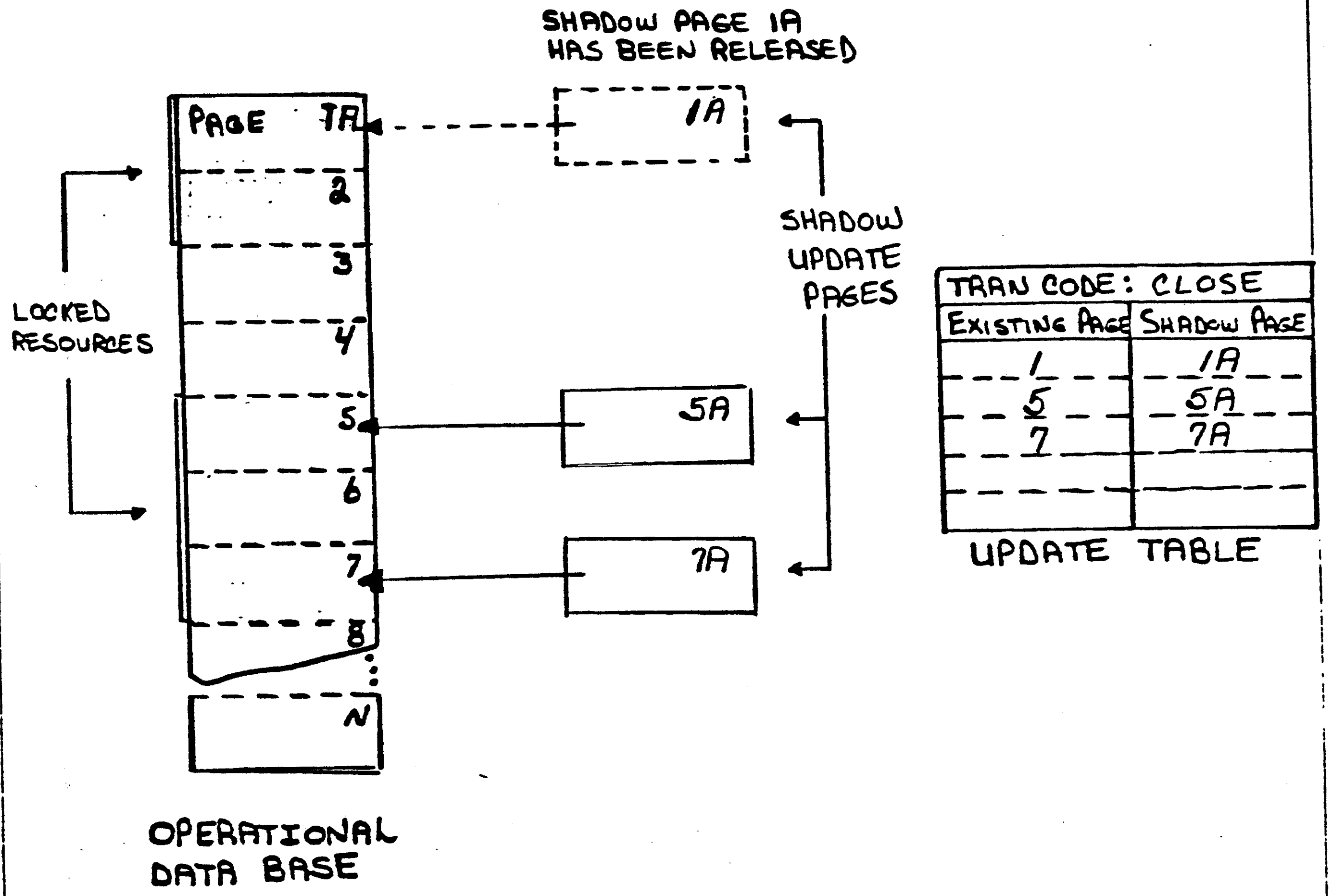


FIGURE II -- RESOURCES LOCKED, TRANSACTION IN PROCESS
SHADOW UPDATE IN PROGRESS



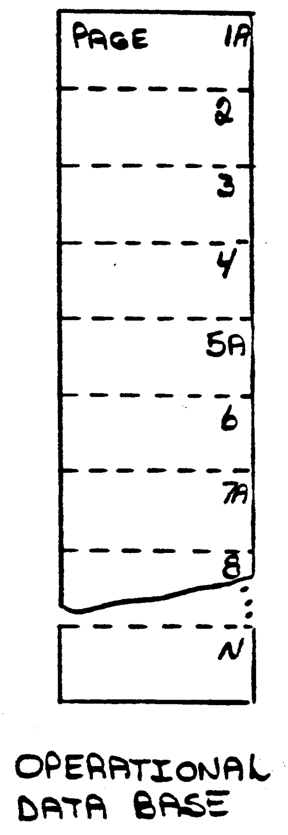
TRAN CODE: CLOSE	
EXISTING PAGE	SHADOW PAGE
1	1A
5	5A
7	7A

FIGURE III -- RESOURCES LOCKED, COMMIT-TRANSACTION HAS BEEN DETECTED, TRANSACTION CLOSED, PHYSICAL DATA BASE UPDATE CAN BEGIN



TRAN CODE: CLOSE	
EXISTING PAGE	SHADOW PAGE
1	1A
5	5A
7	7A

FIGURE IV -- RESOURCES LOCKED, TRANSACTION CLOSED, PHYSICAL DATA BASE UPDATE IN PROGRESS



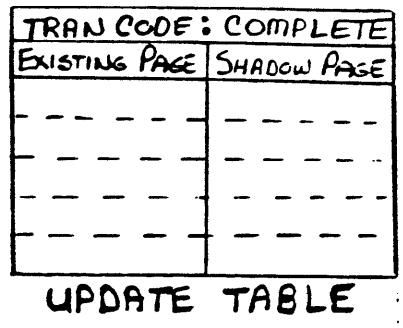


FIGURE V -- RESOURCES UNLOCKED, TRANSACTION COMPLETE, PHYSICAL DATA BASE UPDATE COMPLETE, ALL SHADOW PAGES RELEASED

The shadow update approach insures data base consistency under a variety of circumstances. If the application program detects that a transaction can not or should not be completed prior to executing COMMIT-TRANSACTION, the program need only execute an ABORT-TRANSACTION statement. ABORT-TRANSACTION releases all shadow update pages created by this transaction, clears the Update Table, sets the Transaction Status Code to COMPLETE, and releases all locked resources (in that order). Thus, the data base appears as if the update sequence had never started processing.

If the run-time control system associated with data base processing (Data Base Control System -- DBCS) detects that a program using a data base has terminated abnormally, the DBCS triggers an ABORT-TRANSACTION against the data base. This procedure returns the base to its last consistent state. Access to the base is then terminated as if the program had relinquished control of the base normally.

o Automatic Soft Failure Recovery

Shadow update assures the integrity of the data base in the event of a "soft" system crash. Let us assume a soft failure (no portion of the data base has been physically damaged by disk failure) which occurs while several programs are processing transactions against a data base. The failure has resulted in the abnormal termination of all running programs, and the system has been restarted with a cold start operation.

The first program to access the data base after system restart detects that several transactions were being processed at the time of the failure -- this determination is made from control information similar to that stored in the Update Tables. Those transactions which were IN PROCESS are simply ignored, and their shadow update pages released back to free storage. The data base is in a consistent state (the state before the transaction began) and the interrupted transaction can be re-entered without fear of double update on a partially completed transaction.

Those transactions which were CLOSED at the time of the failure are logically complete, but the actual state of the data base is unknown. It can be assumed that the system failure occurred during the linkage operation which updates the physical data base. CLOSED transactions are simply reprocessed through the linkage procedure. "Existing" pages which have been released are ignored, since they have already been replaced by updated pages. Thus, transactions which are logically complete are processed to physical completion and locked resources are released.

The first program to access the data base after system failure assures the consistency of the entire data base before processing transactions or allowing other programs to access the data base.

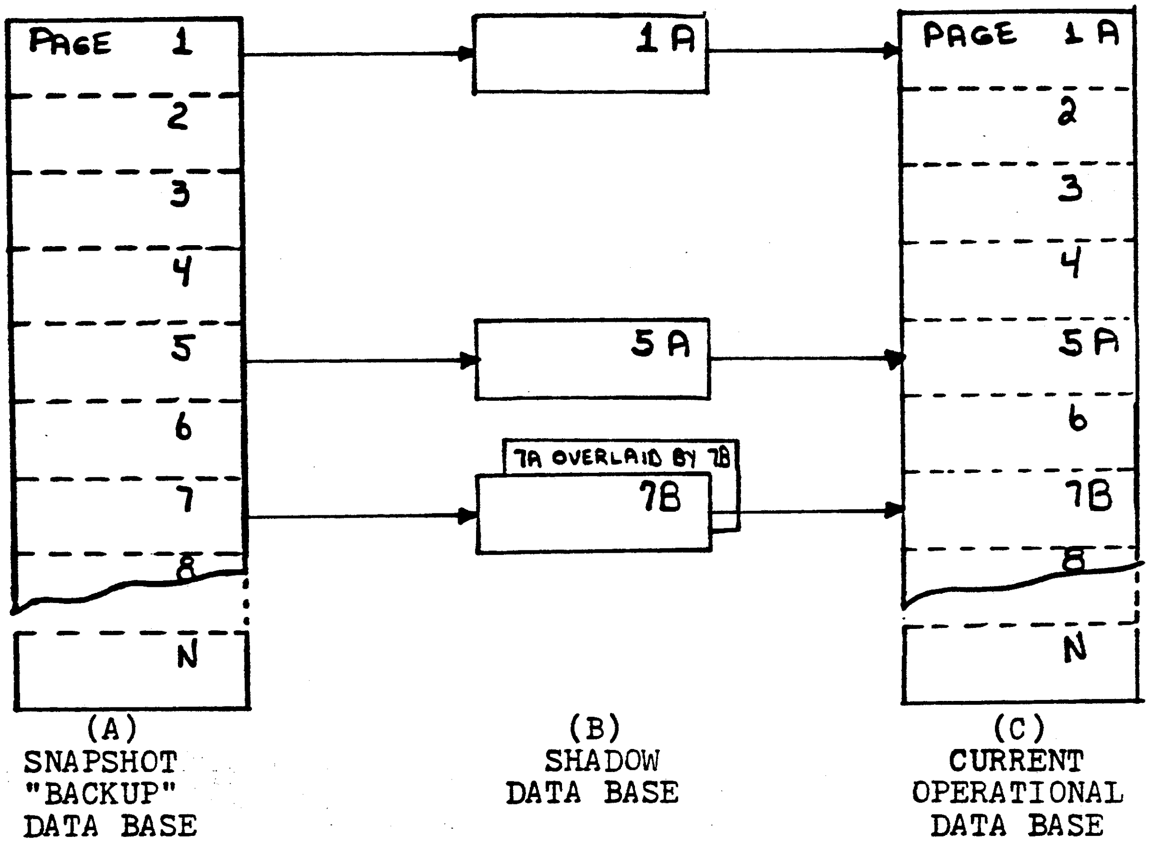
o Hard Failure Integrity Protection/Recovery

DBMS-X is designed to allow efficient, rapid recovery from "hard" system failures. A "hard" system crash is one in which some portion of the physical data base is destroyed. The usual recovery procedure in the event of this type of failure involves restoring the data base from a previous backup and employing a mechanism to roll the data base forward to a point just prior to the hard system failure.

At the user's option, DBMS-X uses a mechanism which applies the shadow update concept to provide a hard failure recovery capability. The user generates a full data base backup for use as a snapshot of the base at a particular point in time. The system then proceeds to post all updates to the data base twice -- once to the operational data base, and once to a shadow copy of the data base which contains only changed data base pages. Over time, the shadow data base contains the most recent copy of all changed data base pages since the backup; if a particular page in the base changes more than once, only the most recent copy of the page resides in the shadow data base. (Figure VI).

In the event of a hard system failure, the backup copy of the data base is restored to the system, and the shadow data base entries are applied against it. At the end of the recovery operation, the data base is as current as the shadow data base.

An interesting feature of this approach to recovery is low overhead, both in disk space and recovery time. This procedure, in conjunction with the journalizing option, offers virtually all of the advantages of before/after image logging without the overhead of



This is the data base which has been backed up -- can be regarded as 'old' data base prior to update	This is the shadow data base which contains only pages which have changed since backup -- note that page 7A has been overlaid by more recent 7B	This is the current data base -- can be viewed as the 'new' data base. It is the same physical base as A with the changes reflected in B applied.

FIGURE VI -- SHADOW DATA BASE CONCEPT AS APPLIED TO HARD FAILURE RECOVERY AND BACKUP

copying physical before and after images. Intermediate data is overlaid (as in pages 7A and 7B in Figure VI) and does not need to be stored or processed as part of the recovery cycle. The DBA may also choose to decrease machine time overhead consumed by data base backup by simply backing up the shadow data base, instead of the entire operational data base.

o Integrity Protection in Program Test Mode

The final major source of data base integrity compromise is the application program. It is extremely important to protect the data base from errors introduced by inadequately tested application programs. Ironically, the very fact that a data base is shared by many users and applications frequently makes it difficult to provide programmers with enough test time to properly test new programs.

DBMS-X uses the shadow data base concept to provide a Program Test Mode facility which allow programmers to test programs against live data bases, without interfering with concurrent production processing.

The execution of a BEGIN-TEST-MODE statement in an application program triggers the establishment of a shadow data base for the test environment. Any changes made to the data base by programs running in test mode are stored in the test shadow data base. The test mode data base remains in existence until specifically destroyed by a programmatic command. Thus, it is possible to run exhaustive test and verification procedures without special control procedures.

DBMS-X provides an extremely high degree of data integrity, at a relatively low cost in storage and processing overhead. System designers can concentrate on application solutions with little concern for the mechanics of data integrity protection.

Capacity to store large data bases:

DBMS-X is capable of supporting large data bases. The BTI 8000 supports an unlimited number of data bases, each with the following size limitations:

Maximum Record Size:	3700 bytes (approx.)
Maximum Page Size:	4096 bytes
Maximum File Size:	67 million records (2^{26})
	2 billion bytes (2^{31})
Maximum Data Base Size:	64 files (areas)
	64 spindles
	4 billion records (2^{32})
	137 billion bytes (2^{37})

Program/Data independence:

A major design goal of data base technology is the establishment and maintenance of data independence. "Logical" data independence allows the overall logical structure of the data base to evolve without forcing changes to application programs which do not use the changed information. "Physical" data independence allows the physical layout and organization of the data to change without affecting the overall logical structure of the data base and/or application programs.

In support of this design objective, DBMX-X functional components can be viewed in terms of the model in Figure VII. The central block represents the overall logical structure of the data base and provides a global view of the entire base. The description of this entity is provided by the SCHEMA. The logical structures depicted on the left of Figure VII are various partial logical views of the data base. These structures are described in SUB-SCHEMAS and are used to provide application programs with specific views of portions of the data base. The actual physical data base is represented on the right side of the diagram.

As implied by Figure VII, the data base management system acts as an interface between physical storage of data and programs. DBMS-X converts the application programmer's view of the data (as described in a subschema) into the overall logical view of the data (as described in the schema) and maps the overall logical view of the data into the physical representation.

An application program which uses the data base involves the following logical entities:

- o The object form of the application
- o The object form of the subschema
- o The object form of the schema
- o The run-time Data Base Control System
- o The physical representation of the data base

Data independence is based upon the ability of each of these entities to change without forcing change in the other entities. The process of linking physical data items with logical data references to those items is called binding. Binding occurs between the schema and subschemas, between application programs and subschemas, and between the schema and the physical data base. In

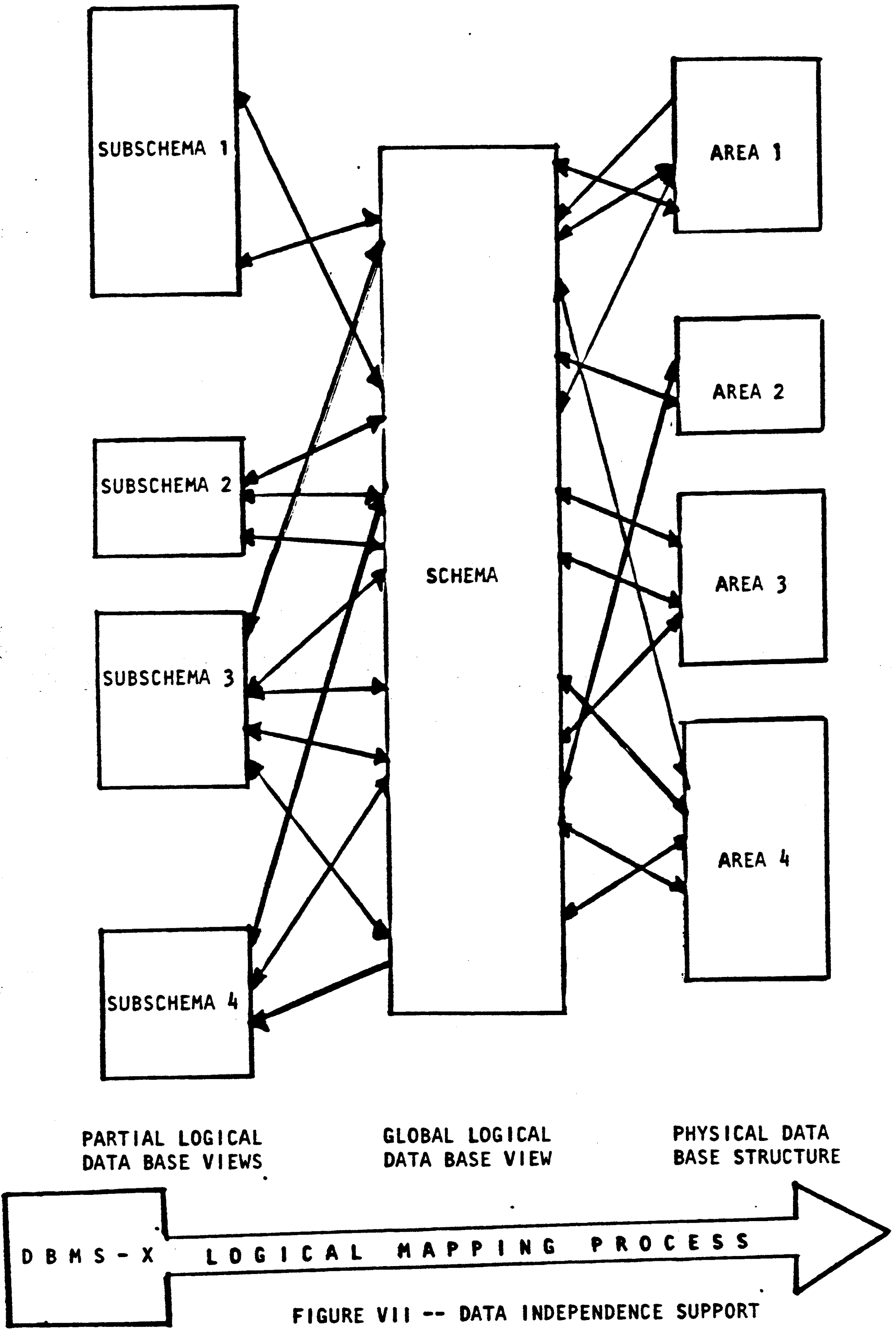


FIGURE VII -- DATA INDEPENDENCE SUPPORT

theory, binding can take place at compilation (note that the schema, subschema and object program are all compiled entities), link edit, program load, or execution time. The closer the binding to execution, the higher the level of data independence between entities.

Subschemas are bound to programs at program compilation. A change in a subschema will therefore require that the subschema and all programs which use the subschema be recompiled. Subschemas are bound to the schema, and the schema bound to the physical data base at program execution. Therefore, changes to the schema have no effect on existing programs and subschemas.

The extent to which DBMS-X achieves data independence can be evaluated by reviewing the support activities which are forced to occur by changes in the overall data processing environment. Figure VIII describes typical events which occur in the program development/ program maintenance environment, and lists the support tasks which must take place with DBMS-X. Note that it is implied that the program which requests a change (as in the changed logical order example) is modified; the first task column refers to other existing programs which use the data base.

DBMS-X eliminates a large portion of the manual overhead required to maintain the correctness of the production processing environment in the face of changing application requirements.

Simple interactive administration:

The data base administration function traditionally has involved several major areas of concern. The Data Base Administrator (DBA)

APPLICATION/PROGRAM CHANGE	--- REQUIRED DATA BASE SUPPORT TASKS ---					
	CHANGE EXISTING PROGRAMS	CHANGE SUBSCHEMA	RECOMPILE EXISTING PROGRAMS	CHANGE CONCEPTUAL SCHEMA	CHANGE PHYSICAL SCHEMA	RESTRUCTURE UTILITY RUN
NEW APPLICATION PROGRAM ADDED -- USES EXISTING DATA	--	--	--	--	--	--
NEW APPLICATION PROGRAM ADDED -- USES CHANGED REPRESENTATION OF EXISTING DATA (e.g. FIXED-POINT INSTEAD OF FLOATING-POINT)	--	YES OR CREATE NEW SUBSCHEMA	--	--	--	--
CHANGE DATA TYPE IN SCHEMA	--	--	--	YES	AUTOMATIC	--
NEW DATA FIELDS ADDED TO SCHEMA	--	--	--	YES	AUTOMATIC	--
NEW RECORD OCCURRENCES ADDED OR OLD RECORD OCCURRENCES DELETED FROM THE DATA BASE	--	--	--	--	--	--
NEW SETS ADDED TO THE DATA BASE	--	YES, NO, OR ADD NEW SUBSCHEMA	YES, IF SCHEMA CHANGES	YES	AUTOMATIC	--
CHANGE LOGICAL ORDER IN A SET	--	--	--	--	--	--
CHANGE PHYSICAL ORDER OF RECORDS IN A SET	--	--	--	--	YES	AUTOMATIC
SPLIT PHYSICAL DATA BASE FILE (AREA) INTO MULTIPLE AREAS	--	--	--	--	YES	AUTOMATIC
DBMS-X SOFTWARE CHANGE (NO INTERNAL FORMAT CHANGE)	--	--	--	--	--	--
DBMS-X SOFTWARE CHANGE (INTERNAL FORMAT CHANGE)	--	--	--	--	--	UTILITY PROVIDED
HARDWARE CHANGE (PERMANENT OR TEMPORARY)	--	--	--	--	--	--

is typically concerned with the following tasks:

- o Logical data base definition and maintenance
- o Physical data base definition and maintenance
- o Performance monitoring

DBMS-X is designed to automate those portions of DBA function which may not require human intervention. Administrative tasks which impact the logical structure of the data base are the primary concern of the DBA. Once the logical data base structure is designed, the schema compiler can choose an appropriate physical implementation. It is not necessary for the DBA to specify initial data base size, bucket sizes, record populations or to reorganize or garbage-collect an existing data base -- the DBCS and the operating system automatically allow for dynamic growth and contraction of the data base.

As the data base evolves over time, the DBA uses simple commands to modify the logical structure of the data base. Most common logical modifications (adding or deleting items, record types, or set types) will not require a reload of the data base.

DBMS-X is designed to guarantee utilization of allocated disk space of at least 50% with utilization levels of better than 90% for growing data bases. Further, search structures are automatically maintained to assure rapid retrieval. The DBA is, however, provided with the ability to monitor performance and adjust the run time environment to improve data base performance.

In short, DBMS-X allows the Data Base Administrator to concentrate on managing the data resource, not the data base.

9.3 Data Definition Language

The Data Definition Language (DDL) is that portion of the DBMS implementation which permits the user to describe and maintain a central description of the contents of the data base. It is used by the Data Base Administrator to create a formal description of the logical data base (Schema), descriptions of various partial logical views of the data base (Subschemas) and to specify parameters which affect physical data base layout and the run-time environment.

Information in the Schema is grouped according to the portion of the logical structure under specification. Each entry type describes a different logical aspect of the data base:

Schema:	The schema entry concerns the entire data base. It specifies the name of the data base and defines the security parameters which control the ability to view and modify the schema itself.
Area:	An area is roughly analogous to a file. The area entry simply names an entity into which records (as specified in Record entries) are stored physically. The area concept deals with the physical storage aspects on the data base and has no impact on the manner in which programs access data; programs interact with the data base on the logical record level.
Record:	A record entry specifies the format -- in

data items and aggregates -- of a record type. The entry names the record type, assigns the area to be used for physical storage of record occurrences, specifies the security controls to be applied against access to record occurrences, and defines the format of the record using COBOL-like syntax. The entry also defines integrity constraints to be applied against data items (fields).

Set: The set entry describes the logical relationships which exist between record types. Specifically, a set type consists of one "OWNER" record type and one or more "MEMBER" record types. Each occurrence of a set must contain an occurrence of its owner record type and any number of occurrences of its member record types.

The set entry names the relationship (set), specifies the record type which owns the set, and describes the security parameters which apply to the set. Optionally, the set entry names a field (or group of fields) which is used to define a particular physical order for the set. Each order parameter triggers the creation of a pointer array for the set, maintained in order by the specified field(s). Pointer arrays provide extremely rapid access to the set in the various specified sequences. Application programs may request access to a set in any order -- those requests which match schema-defined order specifications simply execute more efficiently.

Information concerning set members is located in the member clause of the set entry.

Member:	The member clause is that portion of the set entry which names the member record types in a set. The clause specifies the record type, the applicable security parameters, and a set membership parameter which declares the inclusion of record occurrences in the set to be permanent or transient.
Group:	A group is a reference entity used to build Access Control Lists which are used in the construction of data security sieves. The group entry consists of the group name and a list of the account IDs which comprise the group. Group names are used throughout the schema in the specification of security parameters.

Set relationships within a data base can be described in Bachman diagram form. Such a diagram uses rectangular boxes to represent record types and single direction arrows to represent sets. The owner of the set is at the tail of the arrow, the member record types at the head. Figure I represents the set EMPLOYEE-TIMECARDS, which is composed of one EMPLOYEE record and from zero to many TIMECARD records. Figure II illustrates the EMPLOYEE-TIMECARDS set as it would appear with two member record types.

A sample schema is provided to illustrate the use of the Data Definition Language. The example is a data base to support

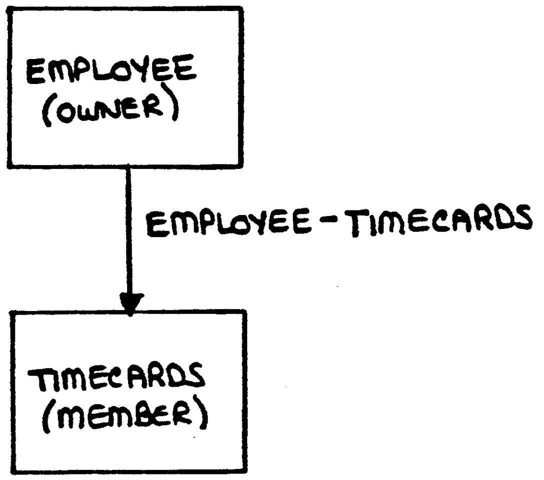


FIGURE I -- EMPLOYEE-TIMECARDS SET
WITH ONE MEMBER RECORD TYPE

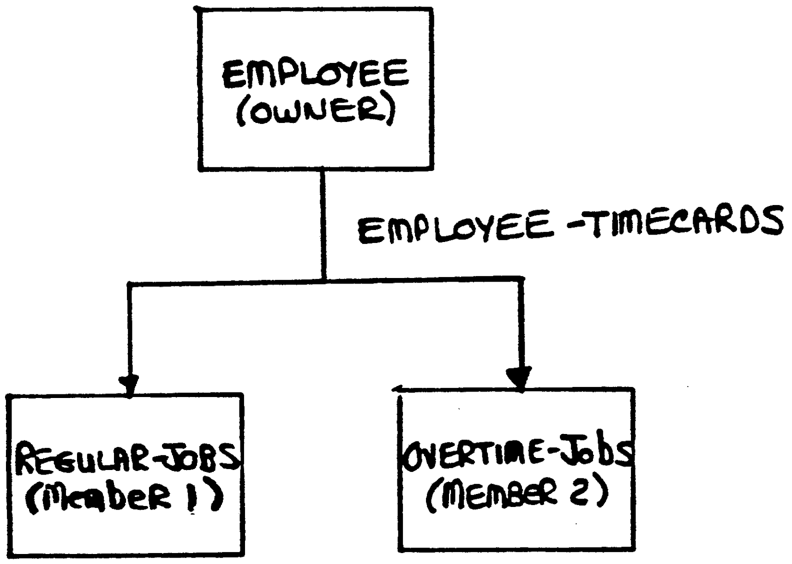


FIGURE II -- EMPLOYEE-TIMECARDS SET
WITH MULTIPLE MEMBER RECORD TYPES

Order Entry, Accounts Receivable, and Inventory Control functions for an organization involved in the distribution industry. For the sake of brevity, the data structures are not as comprehensive as those normally encountered in an operational environment.

The following record types are proposed to support the target functions:

- o CUSTOMER record type -- One record occurrence per customer containing statistical and account information relating to the customer account.

- o ORDER HEADER record type -- One record occurrence per customer order containing static information which applies to the entire order.

- o ORDER LINE ITEM record type -- One record occurrence per order line item consisting of information about the item ordered. A group of ORDER LINE ITEM records will relate to a single order.

- o INVENTORY ITEM record -- One record occurrence per item maintained in inventory, consisting of statistical and accounting information related to the item.

The Bachman diagram for this data base (Figure III) defines the sets which exist among these record types. The Data Definition Language has been used to define the schema (schema name -- DISTRIBUTION) for this data base.

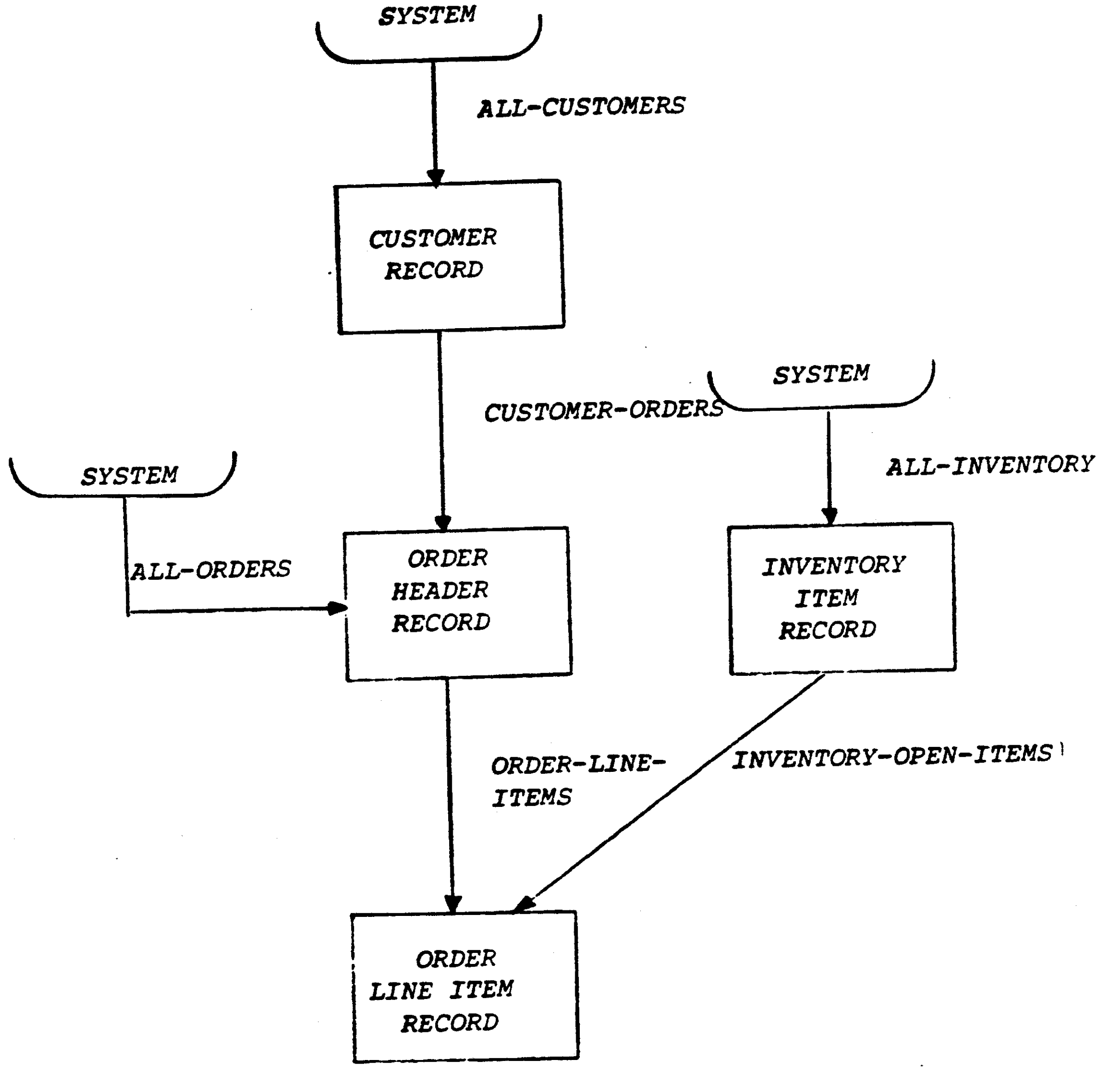


FIGURE III -- BACHMAN DIAGRAM FOR ORDER ENTRY/INVENTORY DATA BASE FOR DISTRIBUTION COMPANY

[illegible]

```
SCHEMA NAME IS DISTRIBUTION
      COMMENT "THIS IS A DATA BASE TO SUPPORT ORDER ENTRY, ACCOUNTS RECEIVABLE,
               AND INVENTORY CONTROL IN A DISTRIBUTION COMPANY ENVIRONMENT."

RECORD NAME IS CUSTOMER-RECORD

      03   CUSTOMER-ID                  TYPE IS INTEGER
      03   CUSTOMER-NAME                TYPE IS STRING 20   CHECK IS ALPHANUMERIC
      03   CUSTOMER-STREET-ADDRESS      TYPE IS STRING 20   CHECK IS ALPHANUMERIC
      03   CUSTOMER-CITY                TYPE IS STRING 15   CHECK IS ALPHANUMERIC
      03   CUSTOMER-STATE               TYPE IS STRING  2   CHECK IS ALPHANUMERIC
      03   CUSTOMER-ZIP-CODE            TYPE IS INTEGER
      03   CUST-ACCTS-REC-BALANCE       TYPE IS INTEGER
      03   CREDIT-LIMIT                 TYPE IS INTEGER
      03   TERRITORY-CODE               TYPE IS INTEGER
      03   SALES-TAX-RATE               TYPE IS INTEGER

RECORD NAME IS ORDER-HEADER-RECORD

      03   CUSTOMER-ID                  TYPE IS INTEGER
             SOURCE IS CUSTOMER-ID OF OWNER OF CUSTOMER-ORDERS
      03   ORDER-NUMBER                 TYPE IS INTEGER
      03   PURCHASE-ORDER-NUMBER        TYPE IS INTEGER
      03   SALES-TAX-AMOUNT             TYPE IS INTEGER
      03   ORDER-DATE                   TYPE IS DATE
      03   INVOICE-DATE                 TYPE IS DATE
      03   GROSS-INVOICE-AMOUNT         TYPE IS INTEGER
      03   NET-INVOICE-AMOUNT           TYPE IS INTEGER
      03   ORDER-BALANCE-DUE            TYPE IS INTEGER

RECORD NAME IS ORDER-LINE-ITEM-RECORD

      03   ORDER-NUMBER                 TYPE IS INTEGER
             SOURCE IS ORDER-NUMBER OF OWNER OF ORDER-LINE-ITEMS
      03   ORDER-LINE-NUMBER            TYPE IS INTEGER
      03   ITEM-IDENTIFIER              TYPE IS INTEGER
             SOURCE IS ITEM-IDENTIFIER OF OWNER OF INVENTORY-OPEN-ITEMS
      03   ORDER-QUANTITY               TYPE IS INTEGER
      03   REQUIRED-DELIVERY-DATE       TYPE IS DATE
      03   SHIPPING-DATE                TYPE IS DATE

RECORD NAME IS INVENTORY-ITEM-RECORD

      03   ITEM-IDENTIFIER              TYPE IS INTEGER
      03   WAREHOUSE-LOCATION-CODE      TYPE IS STRING  5   CHECK IS ALPHANUMERIC
      03   ITEM-DESCRIPTION             TYPE IS STRING 20   CHECK IS ALPHANUMERIC
      03   ITEM-PRICE                   TYPE IS INTEGER
      03   ITEM-DISCOUNT-CODE           TYPE IS INTEGER
      03   QUANTITY-ON-HAND             TYPE IS INTEGER
      03   QUANTITY-ON-ORDER            TYPE IS INTEGER
      03   QUANTITY-ALLOCATED           TYPE IS INTEGER
      03   QUANTITY-AVAILABLE           TYPE IS INTEGER

SET NAME IS ALL-CUSTOMERS
      OWNER NAME IS SYSTEM
      ORDER IS BY CUSTOMER-ID                        DUPLICATES NOT ALLOWED
      MEMBER NAME IS CUSTOMER-RECORD                 PERMANENT

SET NAME IS ALL-INVENTORY
      OWNER NAME IS SYSTEM
      ORDER IS BY ITEM-IDENTIFIER                    DUPLICATES NOT ALLOWED
      MEMBER NAME IS INVENTORY-ITEM-RECORD           PERMANENT

SET NAME IS ALL-ORDERS
      OWNER NAME IS SYSTEM
      ORDER IS BY ORDER-NUMBER                       DUPLICATES NOT ALLOWED
      MEMBER NAME IS ORDER-HEADER-RECORD             PERMANENT

SET NAME IS CUSTOMER-ORDERS
      OWNER NAME IS CUSTOMER-RECORD
      ORDER IS BY ORDER-NUMBER                       DUPLICATES NOT ALLOWED
      MEMBER NAME IS ORDER-HEADER-RECORD             PERMANENT

SET NAME IS ORDER-LINE-ITEMS
      OWNER NAME IS ORDER-HEADER-RECORD
      ORDER IS BY ORDER-LINE-NUMBER                  DUPLICATES NOT ALLOWED
      MEMBER NAME IS ORDER-LINE-ITEM-RECORD

SET NAME IS INVENTORY-OPEN-ITEMS
      OWNER NAME IS INVENTORY-ITEM-RECORD
      ORDER IS BY CUSTOMER-ID, REQUIRED-DELIVERY-DATE  DUPLICATES ALLOWED
      MEMBER NAME IS ORDER-LINE-ITEM-RECORD          TRANSIENT
```

9.4 Data Manipulation Language

The Data Manipulation Language (DML) is that portion of the DBMS-X implementation which is concerned with the interface between the data base and various programming languages. Typically, the DML is a set of extensions to a data base host language which allows application programmers to access the data base using extended host language facilities.

Programs which access DBMS-X can be written in COBOL, FORTRAN, or PASCAL-X. Each host language has been extended to allow data base functions to be accepted directly by the language compiler, eliminating the need for DML preprocessors. The syntax of the data base extension set conforms to the overall form and style of the host language.

DBMS-X supplies the application programmer with 20 data base verbs. The logical function of each verb is outlined below:

START-DBCS	Requests use of DBCS
INVOKE-DBCS	Logical data base OPEN
BEGIN-TEST-MODE	Establishes test shadow data base
END-TEST-MODE	Releases test shadow data base
BEGIN-TRANSACTION	.
ABORT-TRANSACTION	. Data base consistency verbs (Section 9.2)
COMMIT-TRANSACTION	.
FIND	Logical SEEK operation
GET	Logical READ operation
OBTAIN	Combined logical SEEK and READ operation
MODIFY	Logical REWRITE operation

STORE	Logical WRITE operation
ERASE	Logical DELETE operation
CONNECT	Adds a record occurrence to a set
DISCONNECT	Deletes a record occurrence from a set
INTERSECT	Various options define and manipulate groups of records or sets
ORDER	Logical SORT performed on groups of records
SET	Manipulates user-controlled record pointers
RELEASE-SEQUENCE	Releases previously ordered groups of record occurrences
REVOKE-DBCS	Logical data base CLOSE

Application prgrams which use the data base are supplied with a particular view of the data base by a DBA-defined subschema. The subschema describes the portion of the data base available to the program. The available data is then accessed and manipulated through the use of the DML host language extensions.

9.5 Data Base Control System (DBCS)

The Data Base Control System (DBCS) is that portion of the DBMS-X implementation which controls the run-time environment associated with data base processing. This system is charged with the actual translation of logical data requests into physical data access and manipulation sequences.

When an application program requests the use of DBMS-X, a concurrent process (DBCS) is spawned which interacts with the data base and the application program. All program interaction with the data base takes place through the DBCS. Communication between the DBCS and the application process occurs through system-specified .PATH devices. The separation of function inherent in this approach protects the data base from the user, while also protecting non-data base files from the data base software.

In general, the organization of a running data base process can be viewed as in Figure I. The Data Base Control Block, object schema, and data base buffers are shared among all users of a particular data base, to ensure integrity and low system overhead. The application domain and the data base domain communicate through the .PATH devices, and are effectively isolated from each other.

A multi-user data base environment is described in Figure II. Although the DBCS prcesses are self-contained entities, a large portion of the DBCS code and data is automatically shared, decreasing overall processing overhead.

DBCS controls individual processes which use the data base, provides automatic run-time conversion between schema and subschema data items, does structural and logical integrity checking on data base content, implements logging and journalizing, and provides automatic soft failure recovery. It creates a flexible, safe operating environment for the DBMS-X user.

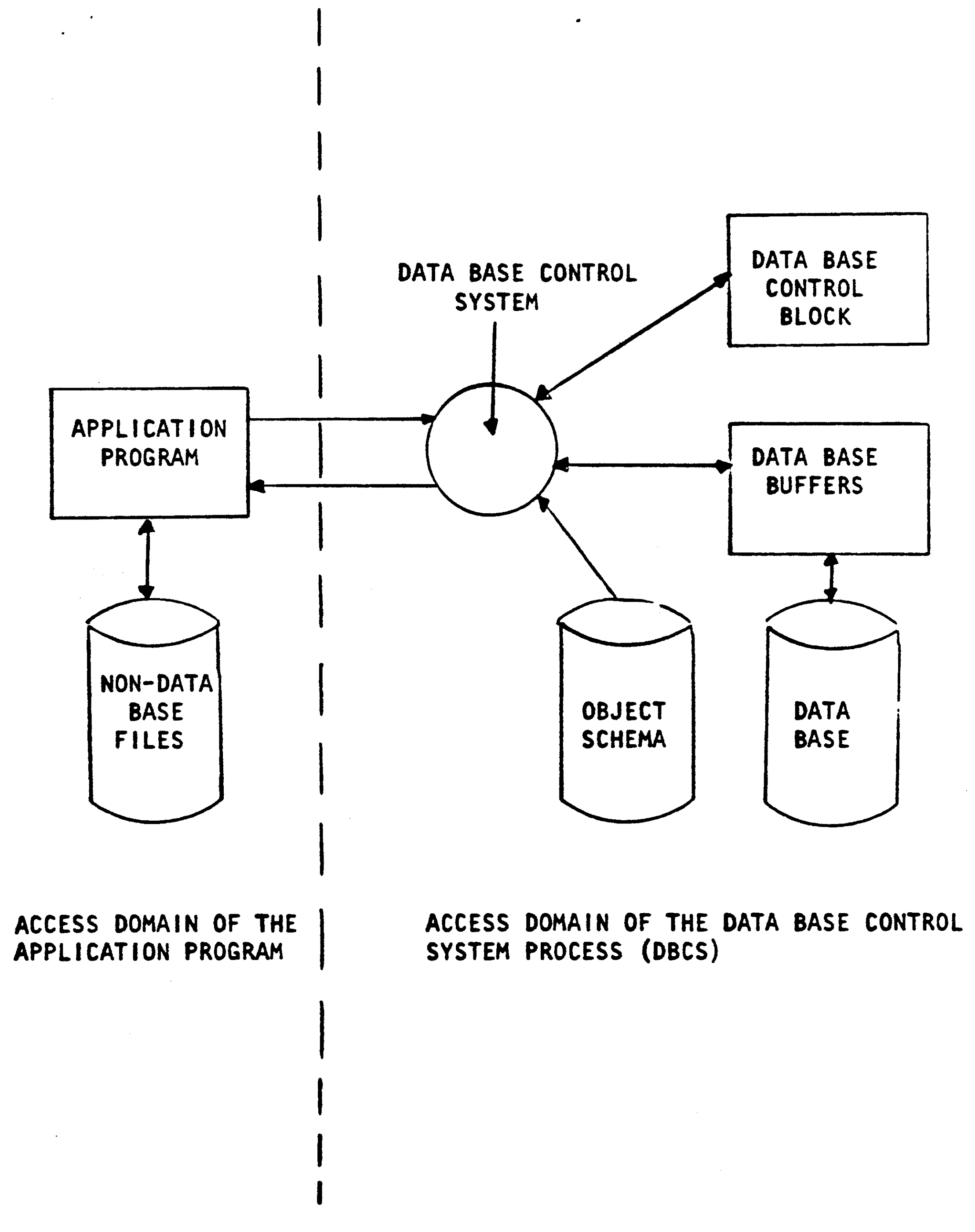


FIGURE 1 -- DATA BASE PROCESS ORGANIZATION

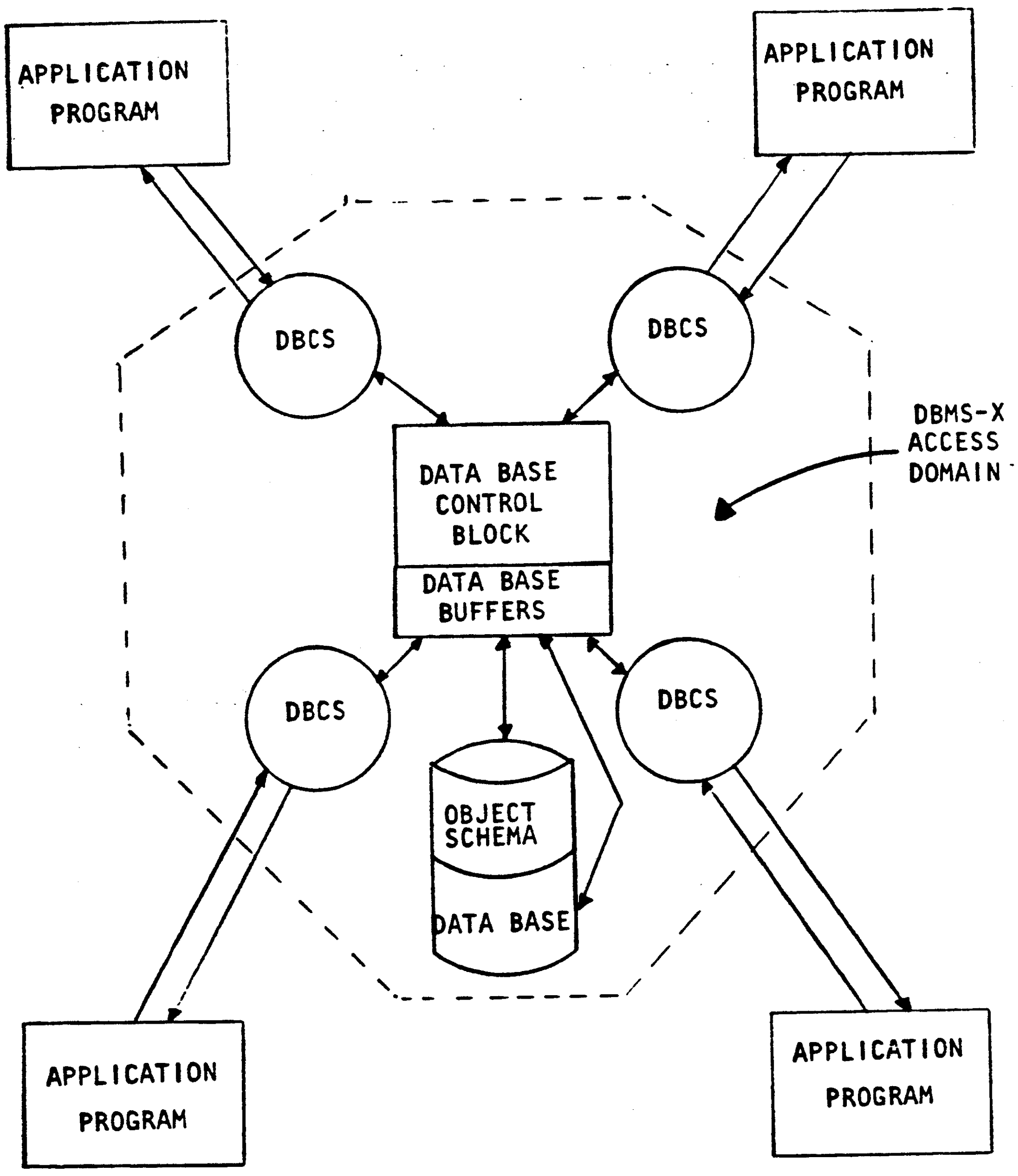


FIGURE 11 -- MULTIPLE USER DBMS-X ENVIRONMENT -- ALL USERS ACCESSING THE SAME DATA BASE

9.6 Data Base Utility Functions

DBMS-X provides the Data Base Administrator with a comprehensive set of utility functions to create, maintain and support evolving data bases. The major functions are listed below:

Data Base Creation:	Compiles schema source; generates object form schema; creates an empty data base. (Note: this process does not require the establishment of pre-allocated files -- all allocation is dynamic.)
Schema Listing Utility:	Provides various schema listings from the object schema; wide range of options, including conceptual schemas, physical schemas, and subschemas.
Journal Log Utility:	Provides a logical log of all data base transactions which change the data base. Intended to be used as application program debugging tool.
Data Base Restructuring Utility:	An interactive utility to aid in the modification of existing data bases. Allows a wide range of restructuring options to be invoked without reorganization of the data base.
Activity Log Utility:	Provides Data Dictionary Information regarding data, schema, and subschema

usage in application systems.
Intended for use as a data administration tool in the evaluation of proposed changes to programs and/or data.

These utility functions are designed to support the effective management of the data resource as a normal byproduct of DBMS-X use.

10. SECURITY

10.1 General

The BTI 8000 has been designed with a commitment to security considerations. The intention of the security mechanisms is to allow the most flexible application of the system that is consistent with full protection of user data, user operation, and system integrity from accident or mischief.

For in-house data processing applications, the system protects operational continuity by insulating its control facilities from user actions, and by conducting its own operations in a crash-resistant manner. Separation of data "ownership" through the account structure automatically limits the damage that can be done by erroneous application program; and, sensitive data can be protected from modification or even examination by those who should not have these privileges (including system operators). The security mechanisms were designed into the system, including its hardware, from the very beginning, and are intended to function in a multi-user on-line environment.

The design also makes the BTI 8000 an excellent system for high-security applications, as well as those in which the system manager and the system users have a fiduciary relationship, such as in a service bureau or a commercial timesharing operation. Managerial accounts can relinquish their privileges to access subordinate accounts. (This is done automatically in the case of proprietary accounts.)

Although physical installation security should not be neglected, the system attends to external protection by automatically encrypting all removeable data storage media, including on-line disk packs and all backup media (packs and tapes).

If desired, the system operator can even disable BTI remote maintenance access, by means of a switch on the operator's panel. Disk packs containing highly sensitive data can be physically removed before re-enabling remote maintenance.

10.2 Account and File Privacy

The account structure is the basic framework for security design, on the grounds that each account normally belongs to one individual; ultimately, security plans enforce privacy, protection, and privilege among individuals, including their actions and the entities they own. All processes, except Monitor processes, run under the auspices of some account, and all data (except spooled data entrusted to the Monitor) belongs to some account.

The account structure is "basically closed and passively secure"; that is, all operations and data remain private within account boundaries unless explicit action is taken to grant foreign access. (Two exceptions to this principle are the public libraries and the managerial hierarchy aspect of the account structure, although implicit access privileges may be revoked in both cases.)

Passwords:

There are no XREQ's to examine any passwords on the system (either account passwords or file shared-list passwords). Internally, all password requirements are stored in encrypted form only; when a password is submitted for access, it is first encrypted, and then the two encrypted forms are compared. There is no way to decrypt the stored passwords, and thus, for increased security, even the system doesn't "know" what they are.

File sharing:

An account can share a file with another account or family of accounts, granting access privileges in one of several ways. The foreign account(s) may have read-only access, even through the owner account retains modification privilege; or the foreign account(s) may have write privilege (possibly limited to "append-only", to establish an inviolate log file); or finally, the foreign account(s) may even be granted privileges over the file equivalent to the owner's. A password requirement may be included, to be satisfied at access time (with the EQUIP).

Since saved logical devices of type .PATH are shared in the same manner as other "files", inter-process communication linkage may also be password-conditioned. When linkage is established via .PATHs, the only view the communicating processes have of one another is that of message senders and receivers, so that mutually suspicious processes may safely communicate.

Managerial privilege:

A managerial account, meaning an account's project master account or division master account, or the system master account, normally has two kinds of privilege to access its subordinate accounts.

First, a manager normally has the same control and access authority over a subordinate account's library that the subordinate account does. One use of this privilege is the installation of a file in the subordinate library, with the file declared "shared" with its owner in a restricted (e.g., read-only) mode.

Second, managers normally have the authority to log into their subordinate accounts without satisfying the password requirement. Of course, the system cannot distinguish among individuals who have access to the same account, so the managers essentially become the account owners when they log in. In particular, this allows the managers to set and change the account's passwords.

Each of the managerial privileges may be forfeited, in cases where the subordinate account must remain private from its managers. Each of these privileges may be disabled only by the manager (forfeited), and re-enabled only by the subordinate account (which would occur if the private account needed "rescuing" by its manager). Since the managers are always in control of their subordinates' use of system resources, forfeiture of access does not imply the possibility of "runaway" accounts.

<u>Encapsulation</u>:

Any account can "encapsulate" itself, and any project or division manager can encapsulate his project or division. When an account or family of accounts is encapsulated, the system constructs a shield around the account(s) that disallows any EQUIP that would permit data to pass out of the encapsulated region.

Encapsulation ensures data privacy in a high-security situation. By encapsulating his family of accounts, a manager prevents any external account from equipping to any file within the encapsulated libraries (except on an append-only basis), even if a subordinate account declares a file shared with an external account. Similarly, an encapsulated account cannot equip for writing any file outside the barrier, even if an external account has granted such permission and informed the encapsulated user.

10.3 Foreign Program Execution

Executable programs, in the form of .CODE files, may be shared by their owning accounts for use by other accounts. Programs may be shared in read-only or execute-only mode, and each page of a program can be protected against modification when running, so that "pure code" segments may be created.

The execution of a borrowed program introduces the possibility of a classic problem in computer security known as the "Trojan Horse". A Trojan Horse is a program innocently borrowed from another owner (who grants access) which, when executed, either steals data (e.g., by piping it back to the foreign account) or does some damage to the environment in which it is running.

The data theft problem is solved by encapsulating the account prior to running a suspect program, as discussed earlier. Damage to the account environment is prevented by means of a series of restriction flags which are part of every account, and which can be set by the account prior to running a suspect program. (The Control Mode RESTRICT and PERMIT commands invoke the XREQ's that set and clear these flags.) When a restriction flag is set, a certain set of Executive Requests are "trapped out": the system will stop a program that attempts to execute an XREQ within the class. The classes of restrictable XREQ's are:

(a) Those which allow catalog information to be fetched (equip to .DIR)

(b) Those which change account limits, such as CPU time, etc.;

(c) Those which set up the program interrupt mechanisms in such a way as to keep all interrupt processing within the program (in particular, so that the program could never be interrupted from the keyboard);

(d) Those which change certain account attributes, including password, Hello and Bye program assignment, and the restriction flags themselves.

A user may also wish to use the restriction flags prior to running one of his own programs, if that program is in the debug stage.

To allow managerial accounts to retain control over their subordinates, .CODE files carry internal restriction override fields to correspond with each restriction flag. Each override field contains the identity of which level of authority (project, division, or system manager) declared the override. Thus a project manager, for example, could prevent a subordinate from invalidating keyboard BREAK control on a given program, if the manager wanted that program to retain control when run by the subordinate. The appropriate override authority fields are cleared, incidentally, when a program is borrowed across project or division boundaries.

10.4 Manager and Operator Privileges

The system recognizes two special classes of Executive Requests: Manager requests, which perform account creation and control functions, and operator requests, which perform system operator functions. The system employs the following safeguards to control the use and propagation of manager and operator XREQ's:

Manager requests:

(a) The XREQ MBLESSP, when executed, grants a "target" program (.CODE file) the authority to execute manager requests (provided certain other conditions hold, as discussed in (c) below). MBLESSP itself will execute if and only if the following two conditions are both met:

(1) The program that attempts the MBLESSP has been granted the authority to execute manager requests; and

(2) The account running this program has the authority to execute the MBLESSP. (This authority is implicit in MASTER accounts, and may be explicitly granted to other accounts.)

Note: BTI supplies an initial "Manager Program" in MASTER.SYS which has the authority to execute manager requests.

(b) The XREQ MBLESSA, when executed, grants a "target" account the authority to execute the MBLESSP XREQ. MBLESSA will execute if and only if all of the following three conditions are met:

(1) The account running the program containing the MBLESSA XREQ is a MASTER account (so that these authority-granting privileges cannot be propagated indiscriminately);

(2) The program containing the MBLESSA has been granted the authority to execute manager requests; and

(3) The target account is a subordinate of the MASTER account running the program.

(c) Finally, a program may execute other manager requests (other than MBLESSP and MBLESSA) if and only if the following two conditions are both met:

(1) The program has been granted the authority to execute manager requests (i.e., it has been the target of an MBLESSP); and

(2) The account running the program has the authority to execute manager requests, either because it is a MASTER account or because it has been the target of an MBLESSA.

Operator requests:

(a) The XREQ OBLESSP, when executed, grants a "target" program the authority to execute operator requests (provided certain other conditions hold, as discussed in (c) below). OBLESSP itself will execute if and only if the following two conditions are both met:

(1) The program that attempts the OBLESSP has been granted the authority to execute operator requests; and

(2) The account running this program has the authority to execute the OBLESSP. (This authority is implicit in the account MASTER. SYS, and may be explicitly granted by MASTER. SYS to other accounts within division .SYS.)

Note: BTI supplies an initial "Operator Program" in MASTER. SYS which has the authority to execute operator requests.

(b) The XREQ OBLESSA, when executed, grants a "target" account the authority to execute the OBLESSP XREQ. OBLESSA will execute if and only if all of the following three conditions are met:

(1) The account running the program containing the OBLESSA XREQ is MASTER.SYS;

(2) The program containing the OBLESSA has been granted the authority to execute operator requests; and

(3) The target account is within division .SYS.

(c) Finally, a program may execute other operator requests (other than OBLESSP and OBLESSA) if and only if the following two conditions are met:

(1) The program has been granted the authority to execute operator requests (i.e., it has been the target of

an OBLESSP); and

(2) The account running the program has the authority to execute operator requests, either because it is MASTER. SYS, or because it is another account within division .SYS that has been the target of an OBLESSA.

APPENDIX A: SUMMARY OF USER-MODE CPU INSTRUCTIONS

A.1 Fixed Point Arithmetic

ADD: operand added to contents of specified register, result stored back in that register

ADDM: ("add to memory") as above, but result replaces operand instead of register

ADDB: ("add to both") as in ADDM, but result also stored in register

ADD2, ADD2M, ADD2B: double-word analogs of above

SUB: operand subtracted from contents of specified register, result stored back in that register

SUBM, SUB2, SUB2M: see ADD family

RSB: ("reverse subtract") contents of specified register subtracted from operand, result stored back in that register

RSBM, RSB2, RSB2M: see SUB family

MUL, MULM, MUL2, MUL2M: multiply family (see ADD, SUB)

DIV, DIVM, DIV2, DIV2M: divide family

RDV, RDVM, RDV2, RDV2M: reverse divide family

LD, LDN (N="negate"), LD2, LDN2: load register family

INCL, INCL2: increment operand by 1, then load reg. with this new value

ST, ST2: store register (single, double)

STW, STW2, STMW, STMW2: store the value "one" (W) or "minus one" (MW)

STU, STU2: store the value "undefined" (hexadecimal 80000000)

STZ, STZ2: store the value "zero"

EXCH, EXCH2: exchange register, operand

INC, INC2, DEC, DEC2: increment/decrement operand by one

INCP, DECP: increment/decrement pointer. These instructions assume the operand is a pointer. The bit length of the pointed-to entity (carried in the pointer) is added to/subtracted from its bit address, thus moving the pointer forward/backward one entry, no matter what the size of the entry.

A.2 Floating Point Arithmetic

These instructions deal with 64-bit (double word) floating-point operands, which have 11-bit biased exponents and 52-bit mantissas. Double-precision floating-point operands (128 bits) are generated and manipulated by software.

FAD, FADM, FADB: floating add ("to memory", "to both")
FSB, FSBM, FMU, FMUM, FDV, FDVM: floating subtract, multiply, divide
FRSB, FRSBM, FRDV, FRDVM: floating reverse subtract, reverse divide
FINC, FDEC: floating increment, decrement memory (by one)
FINCL: increment floating-point operand by 1, then load adjacent registers with this new value

A.3 Boolean Arithmetic

AND, ANDM, AND2: similar to fixed-point ADD family
BSUB, BSUBM: result = register AND NOT operand (Boolean subtract)
BRSBM: Boolean reverse subtract to memory
IOR, IORM, IOR2: inclusive OR family
XOR, XORM, XOR2: exclusive OR family

SETT: (set and test) set operand to one after setting condition bits to comparison of register and operand (used for locking of critical regions)

A.4 Jumps

Unconditional: JMP (load Program Counter with operand)

Conditioned on PSR condition bits: JCC,JCS (if carry clear/set), JOC, JOS (if overflow clear/set), JEQ, JNE, JLT, JGT, JLE, JGE

Conditioned on comparison of register contents to zero ("Z") or minus one ("MW"): JEQZ, JEQZ2, JNEZ, JNEZ2, JLTZ, JLTZ2, JGTZ, JGTZ2, JLEZ, JLEZ2, JGEZ, JGEZ2, JEQMW, JNEMW

Bit tests: JBT, JBF (if bit in register true/false)

Address tests: JZA, JNZA (if address field of register zero/non-zero)

Register increment/decrement: IRJ, DRJ (inc/dec register, then jump if result not equal to zero); JIR, JDR (if register not equal to zero, inc/dec register and jump)

Linkage jumps, conditioned on zero/non-zero address field fetched through register: LJZA, LJNA (load register with address field of word it points to, then jump if result zero/non-zero); RLJZA, RLJNA (remember, link, and jump -- save register in adjacent register, then proceed as in LJZA, LJNA)

A.5 Subroutine Linkage

Several instructions are provided for subroutine linkage; they check entry points and provide parameter type-checking for the subroutine. The calling sequence and the entry sequence are executed part by part, passing one parameter at a time with the PAR (pass parameter) instructions on the calling side and corresponding STP (store parameter) instructions on the subprogram side. These instructions specify the parameter type (including "2" for doubleword), whether the parameter is being passes by location or value ("V"), and whether this is the last ("L") parameter in the protocol.

CALL, CALLNP ("NP" = no parameters): begin linkage from calling side
ENTR, ENTRNP, ENTRS ("S" = start, for non-standard parameter passing): begin subroutine
PAR, PAR2, PARL, PAR2L, PARV, PARV2, PARVL, PARV2L: pass parameter
STP, STP2, STPL, STP2L, STPV, STPV2, STPVL, STPV2L: store parameter
LEAVE: leave subroutine
LDPC, LDPCS: load Program Counter ("S" = also load status bits)
EXPC, EXPCS: exchange Program Counter (and status) with operand
JSR: jump and save return address in register

A.6 Compare Instructions

CKB, CKB2, FPCKB, I2CKB: bounds checking for array indexing
CPR, CPR2, UCPR, UCPR2: signed/unsigned compare register with operand

MCPR: masked compare register with operand (adjacent register selects bits)
CMZ, CMZ2: compare operand ("memory") to zero
STLEQ: store logical one ("1") iff condition bits = "EQ", else store zero
STLNE, STLLT, STLGT, STLLE, STLGE: as above for other conditions

A.7 Character Instructions

These instructions are interruptible, and deal with character strings whose starting address and length are given by register values. The CMOVE instruction loads and stores whole words and thus is quite efficient no matter what the character alignment might be.

CSRCH: search for a specified character in a specified string
CMS: compare strings (can be paired with CSRCH to search for substrings)
CMOVE: move string

A.8 Miscellaneous Instructions

LDPSR, STPSR: load/store Process Status Register
CLPSR, IORPSR, XORPSR: PSR bit manipulation
HIB, HIB2: find location of leftmost one-bit in operand
LEA, LEA2: load effective address (generate a pointer)
XCT: execute operand as if it were an instruction (one level only)

LSRCH: linked list search. Searches through a linked list of structures for a match between the value in a specified part of each structure and a value in a register (or register pair)

PMUT: (permute) Using a 32-word table, this instruction can permute bits in a register, encrypt data, compute parity, and form block checksums.

NOP: no operation

A.9 Address Modes

In addition to specifying a register, many instructions also specify an operand through an address mode field. Address mode parameters can in turn involve the specification of one or two registers used to arrive at an operand. Indirect addressing proceeds through "pointers", which themselves specify five different methods of addressing. The following summary is by class, with the number in parentheses representing the total number of modes in each major class. The distinction between single-word and double-word addressing (for word-size operands) is not considered in this count, since that distinction is made in the instruction operation-code field.

(1) DIRECT
(1) INDEXED
(3) IMMEDIATE
(5) INDIRECT
(2) INDIRECT AND INDEXED (first indirect, then indexed)

(1) REG1 (register select, with value biased)
(1) ARWD1 (offset from base register)
(1) CACH1 (offset to character from base register)
(5) FPVR1 (offset from base register, then indirect)
(1) REG2 (as in REG1, but indexed)
(1) ARWD2 (offset from base register, then indexed)
(1) CACH2 (offset from base register, then indexed to character)
(2) FPVR2 (offset from base register, then indirect, then indexed)
(1) CBM (circular bit-string mode)
(1) ZBM (zig-zag bit-string mode)
(1) STK (stack mode)
(4) TCONV (type conversions: integer/floating-point, etc.)

Totals: 32 address modes through 17 classes

APPENDIX B: MAINTENANCE AND SUPPORT

B.1 Self-test

The detection of system failure, as discussed earlier, is perhaps the most critical function in maintaining the correctness and usability of a computer system. Malfunctioning components should be shut down immediately to prevent proliferation of erroneous data items or invalid system structures. If self-consistency checking modules in the hardware or operating system cause the system to halt, the next step is the isolation of the failure.

Hardware checking:

Pressing the start key on the operator's panel causes the SSU to send a self-test command signal down the VRA bus. Upon receipt of the signal, every module on the bus, plus the controllers connected to the PPU channels, runs a self-test diagnostic exercise. All results are sent back to the SSU, which identifies the failed units on the operator's display. As a further aid to locating the problem, a module which fails its test will light a small red light on its board, so the operator can easily find the proper board to remove.

Software checking:

With different settings of the front panel switches, the system operator can cause software diagnostics to run. Some of these validate disk structures, many of which can be rebuilt automatically from redundant information. A memory dump routine writes the contents of in-core Monitor tables and code areas to a reserved location on the system disk volume, for later investigation by BTI remote maintenance.

B.2 On-line System Support

BTI maintains a National Service Center for on-line support of BTI systems. A BTI 8000 at the service center is equipped with automatic telephone dialing equipment connected to its ports, and can automatically dial into the Remote Front Panel facility on all BTI 8000 systems. (The remote system must have remote maintenance enabled on its front panel, and must have an automatic-answer modem connected to its lowest-numbered asynchronous communications port.)

The service computer can run remote diagnostics in more detail than is available through the operator's panel, and can investigate system structures and memory dump areas. From a terminal connected to the system, a BTI service engineer can in many cases patch structures to salvage user files or processes that might otherwise be lost.

B.3 Hardware Support

If diagnosis indicates the need for a replacement hardware module, one will be shipped from the nearest BTI parts depot, with a regional service engineer dispatched if necessary. (In a redundant configuration, the system will be operational while awaiting hardware replacement.) Most module replacements can be performed by customer personnel.

A "hot spares" policy reduces the time required for arrival of a major module (CPU, MCU, PPU, SSU) to literally minutes. If there are unused slots on the VRA bus, BTI can supply extra major modules on a "hot spare" basis. For a monthly charge as opposed to an outright purchase, the modules are installed in spare bus slots in a non-operational mode; they are electrically powered (hence "hot"), but are not functioning parts of the system. Such modules can be regarded as ready-to-use spare parts in a special parts depot which is actually on the customer site; the monthly charge is therefore a premium service charge for the establishment of this "depot".

B.4 Software Support

The BTI National Service Center maintains a round-the-clock service "hotline" for servicing hardware and Monitor problems. Problems with BTI-supplied "user-mode" (non-Monitor) software, such as compilers and utilities, should be referred to the regional systems analyst assigned to the installation. If the analyst cannot resolve the difficulty directly, he will refer the problem to the home office Software Support Section, available during normal West Coast business hours. With customer permission, Software Support personnel log on to the system in question, to re-create and isolate the problem; in doing so, they use a non-privileged user account.

B.5 Training and Documentation

BTI currently offers a two-week introductory course on the BTI 8000 system, including a system overview, system management, and system operations. A one-week course is also offered on DBMS-X. Other classes are currently held on request.

In addition to this manual, the following technical literature on the BTI 8000 is available:

Virtual Machine Functional Specifications
Manager Manual
Operator Manual
Control Mode
Editor
Loader (series, including compiler-writers' information)
Assembler
Sort/Merge
Copy

languages: PASCAL-X; COBOL; FORTRAN; RPG II; BASIC-X

DBMS-X (series of manuals)
Asynchronous Port Interface Guide
Configuration Guide
Security Handbook
Site Preparation Guide
Hardware Service Notes (series)

BTI 8000 Technical Bulletins are also published and distributed as required.

APPENDIX C: ABOUT BTI

Starting as a commercial timesharing service (with the name Basic Timesharing, Inc.) in 1968, BTI evolved into a developer and manufacturer of timeshared computer systems. Since its first deliveries in 1971, BTI has delivered over 700 Model 3000, 4000, and 5000 series systems, with installations in over 40 states in the U.S., in Canada, and in Europe. These systems are used in a variety of applications, including general accounting, inventory control, timesharing services, product testing, research and development, medical laboratories, health care systems, and school administration. The new BTI 8000 system is a planned outgrowth of BTI's special experience in the design, manufacture, and support of multi-user interactive computer systems.

Regional Sales Offices	East:	3 Executive Campus, Cherry Hill, NJ 08002 (609) 662-1122
	South:	1545 West Mockingbird Lane, Dallas, TX 75235 (214) 630-2431
	Midwest:	3001 Metro Drive, Minneapolis, MN 55420 (612) 854-1122
	West:	870 West Maude Avenue, Sunnyvale, CA 94086 (408) 733-1122
	United Kingdom:	Brunel Way, Slough SL1 1XN Berkshire, 0753 70044

BTI COMPUTER SYSTEMS

870 West Maude Avenue, Sunnyvale, California 94086 (408) 733-1122

Printed in U.S.A.