Proceedings of the

WESTERN JOINT COMPUTER CONFERENCE

February 26-28, 1957 Los Angeles, Calif.

Sponsors:

THE INSTITUTE OF RADIO ENGINEERS

Professional Group on Electronic Computers

THE AMERICAN INSTITUTE OF ELECTRICAL ENGINEERS

Committee on Computing Devices

THE ASSOCIATION FOR COMPUTING MACHINERY

Printed in the United States of America

Price $4.00

PROCEEDINGS OF THE WESTERN JOINT COMPUTER CONFERENCE

PAPERS PRESENTED AT
THE JOINT IRE-AIEE-ACM COMPUTER CONFERENCE
LOS ANGELES, CALIF., FEBRUARY 26-28, 1957

Sponsors
THE INSTITUTE OF RADIO ENGINEERS
Professional Group on Electronic Computers
THE AMERICAN INSTITUTE OF ELECTRICAL ENGINEERS
Committee on Computing Devices
THE ASSOCIATION FOR COMPUTING MACHINERY

Published by
The Institute of Radio Engineers
1 East 79th Street, New York 21, N. Y.
for the
Joint Computer Committee

ADDITIONAL COPIES

Additional copies may be purchased from the following sponsoring societies at \$4.00 per copy. Checks should be made payable to any one of the following societies:

INSTITUTE OF RADIO ENGINEERS
1 East 79th Street, New York 21, N. Y.

AMERICAN INSTITUTE OF ELECTRICAL ENGINEERS
33 West 39th Street, New York 18, N. Y.

ASSOCIATION FOR COMPUTING MACHINERY
2 East 63rd Street, New York 21, N. Y.

LIST OF EXHIBITORS

ALADDIN RADIO INDUSTRIES, INC.	Los Angeles, Calif.
AMP, INC.	Harrisburg, Pa.
AUTONETICS	Downey, Calif.
BENDIX COMPUTER DIVISION	Los Angeles, Calif.
BERKELEY DIVISION OF BECKMAN INSTRUMENTS INC.	Richmond, Calif
C. P. CLARE & CO.	Chicago, Ill.
COLEMAN ENGINEERING CO., INC.	Los Angeles, Calif.
COMPUTER CONTROL CO., INC.	Wellesley, Calif.
ELECTRODATA DIVISION OF BURROUGHS CORP.	Pasadena, Calif.
ELECTRONIC ASSOCIATES	Long Branch, N. J.
ELECTRONIC ENGINEERING CO. OF CALIF.	Los Angeles, Calif.
ENCYCLOPEDIA BRITANNICA	Los Angeles, Calif.
ENGINEERED ELECTRONICS CO.	Santa Ana, Calif.
FERRANTI ELECTRIC, INC.	Brooklyn, N. Y.
FRIDEN CALCULATING MACHINE CO., INC.	San Leandro, Calif.
G. M. GIANNINI & CO., INC.	Monrovia, Calif.
HUGHES AIRCRAFT CO.	Culver City, Calif.
INTERNATIONAL BUSINESS MACHINES CORP.	New York, N. Y.
LIBRASCOPE, INC.	Glendale, Calif.
F. L. MOSELEY CO.	Pasadena, Calif.
G. E. MOXON SALES	San Mateo, Calif.
NORDEN-KETAY CORP.	Gardena, Calif.
PERLMUTH INSTRUMENTS ENTERPRISES	Los Angeles, Calif.
RAMO-WOOLDRIDGE CORP.	Los Angeles, Calif.
J. B. REA COMPANY, INC.	Santa Monica, Calif.
REMINGTON-RAND	New York, N. Y.
T. LOUIS SNITZER CO.	Los Angeles, Calif.
SOROBAN ENGINEERING, INC.	Melbourne, Fla.
SPRAGUE ELECTRIC CO.	North Adams, Mass.
STROMBERG CARLSON	San Diego, Calif.
TELEMETER MAGNETICS, INC.	Los Angeles, Calif.
UNDERWOOD CORP.	Long Island City, N. Y.
JOHN WILEY AND SONS, INC.	New York, N. Y.

WESTERN JOINT COMPUTER CONFERENCE COMMITTEE

MEMORIAL TO JOHN VON NEUMANN

The 1957 Western Joint Computer Conference herewith honors the memory of Dr. John Von Neumann. One of the world's greatest mathematicians in our age, Johnny, as we admiringly called him, made many basic contributions to both the theory and practice of electronic digital computers. While we are sad when we think of the tragic loss from his early death, we recall with much pleasure the brilliant and beautiful example which he set us in his work. Let us emulate his humble behavior, his clear logic, and his deep penetration as we carry on the important work on automatic computation.

FOREWORD

For a number of years the East and West Coasts have been the setting for Joint Computer Conferences, sponsored by the Institute of Radio Engineers, the American Institute of Electrical Engineers, and the Association for Computing Machinery, where designers, users, and other interested personnel exchange information on electronic computing equipment. As in the past, a balance between analog and digital computing techniques is maintained. The theme this year is

"Techniques for Reliability."

JOHN L. BARNES
Conference Chairman

TABLE OF CONTENTS

Introductory Remarks

EDWARD P. COLEMAN†

WHAT does reliability mean? It is not strange to find that the term "reliability" means many things to many people. However, we hope to illuminate a number of these meanings which relate to the art, science, and industry of computing. Typical of many definitions in use today is the following:

> "*Reliability* is the *probability* of a *system* performing its purpose *adequately* for the *period of time* intended under the *environmental conditions* encountered."

In introducing the subject, one should speak briefly of some of the past and present trends in reliability. First, we mention the concept of improvement of reliability by the detection of unreliability. In order to isolate, examine, and improve reliability of a system, the reliability engineer puts his best efforts on the unreliability problem. He studies the failures in the system for it is only through corrective action on failed elements in a given system that significant improvement can be made. This technique is an old problem to quality control engineers, who have worked out many standard procedures for detecting unreliability based upon the Shewhart Control Chart and other fundamental contributions of the last quarter of a century.

A second concept, which is almost an economic derivative of the first, is that of improvement of reliability by the prevention of unreliability. Significant advances in reliability procedures are being made today, many of which have as their underlying principle the prevention of unreliability before hardware is put into production.

The placement of emphasis on unreliability appears to be a negative approach, which is standard practice in quality control organizations and which uses this so-called negative approach. In the quality control division of manufacturing industries, parts may be classified as "defective" or "nondefective." At the end of any such inspection, the number of defective parts are counted. If the number of defective parts exceeds a predetermined allowable number, the production process is halted, and it may not be resumed until the assignable cause for defective products is found and removed. Thus, the tradition in quality control of "detecting defects" and "preventing defects" seems to have a continued longevity in modern reliability techniques.

This point suggests what might appear to be a paradox. As an organization approaches its objective of the total prevention defects, it would appear to have less and less work to do in the future and ultimately none at all. This kind of thinking has manifested itself in industrial organizations in many forms. It has caused some quality administrators to not proceed first directly to the most important reliability problems. Moreover, it has caused some to attempt to build beautiful and permanent procedures for processing unreliability information. One moment of reasoning will show that reliability engineers are needed most where the going is most difficult and where reliability is least predictable. It goes against better nature to leave a beautiful, consistent, and predictable process with little or no unreliability and proceed to one which is ugly, inconsistent, and unpredictable; but this is the lot of the modern reliability engineer.

There are many terms being used today in reliability considerations. Let us list a few of these:

Physical Terms—Part, item, subassembly, assembly, and system.

Merit Terms—The term reliability itself as applied to general effectiveness of system. Reliability in supporting equipment and in operations. Minimum acceptable reliability and mean-time to failure.

Mathematical Terms—Risk, hypothesis, test, random variable, probability, population parameter, sample, and statistic.

Acceptance and Control Terms—Quality characteristic, rational subgroup, attributes, variables, process average quality, sampling plan, sample size, and operating characteristics function.

We first turn our attention to the fundamental concepts of reliability and then to the various details of the problem.

† Univ. of Calif., Los Angeles, Calif.

Keynote Address—Techniques for Reliability in Computers for Weapon Control

JAMES M. BRIDGES†

THE RAPID advances made in computer developments during the past few years have had a profound effect upon the security and economy of the country and upon all our lives. Today, the influence of the high-speed, high-capacity computing machine is being felt throughout our total society: in industry, commerce, science, education, medicine, and in many other areas of human existence and progress. The most significant use of the computer, however, in this era of international instability, is its vital role in maintaining our national security.

Because of my association with the Department of Defense, I am naturally most interested in those computer applications which are of the greatest importance to our defense. I wish I could discuss in detail all the different ways in which various kinds of computing machines are being used throughout the military organization. Since that would not be appropriate here, I am going to limit my remarks to the types of computers used for the dynamic control of weapons and weapons systems.

Since the computer is now essential to the effective performance of all modern weapons and weapons systems, it is obvious that a very high level of reliability is essential. I can assure you that we in the Department of Defense consider that the theme "Techniques for Reliability" is completely appropriate for this Joint Computer Conference.

I shall begin my discussion by presenting a little more detail on the widespread usage of computers in weapon control, together with a few highlights of their developmental history. Perhaps I should make it clear at this point that I use the expression "weapons and weapons system control" to include all computers involved in direct control of weapons such as guns, missiles, torpedoes, rockets, bombs, or aircraft and those involved in such functions as tracking, threat evaluation, and weapon assignment.

Although computing machines have received much publicity over the past few years, I seriously doubt that the vital role they have played in the development of military weapons is generally appreciated.

It is probably not widely known that the fire of naval and army artillery was being controlled with computing devices even before World War I started. I doubt if many appreciate the fact that the precision and capabilities of these weapon control computers have advanced steadily since Hannibal Ford started development of his first computer for naval fire control in 1915, until today practically every offensive or defensive weapon depends for its effective operation upon one or more of these computing devices, some very simple and others even more complex than the largest machines in commercial use today.

† Office of the Assistant Secretary of Defense, Washington, D. C.

On one end of the size-complexity scale is the tiny computer that is packed into the nose of a medium-caliber bullet to compute the point in space with respect to an air target at which detonation should occur. On the other end of this scale are the huge digital computers in the ground environment of the air defense system, which employ tens of thousands of electron tubes and occupy thousands of square feet of floor space. Between these two extremes of size and complexity are scores of different kinds and sizes of computers, each performing a specific function in the dynamic control of some weapon or weapons system. Although the performance, complexity, and packaging requirements of these many types of control computers differ widely, the need for a high degree of precision and operating reliability is common to all.

Until very recently, all these diversified weapon control computers were of the analog type. Although much development work has been done on digital weapon control computers, to my knowledge there is no digital weapon control computer in actual military service operation.

Because the history of weapon control is truly the history of analog computer development, it may be of interest to review very briefly some of the development highlights. As I mentioned before, the history of the fire-control computer in this country started in 1915 when Hannibal Ford began to develop the first computer to control naval surface-to-surface guns. His early computers, known as "rangekeepers," represented the first application of precision analog techniques to the solution of the gun fire-control problem.

At the conclusion of World War I, the need for control of surface guns against aircraft became apparent, and Ford again pioneered with the development of the first antiaircraft-gun fire-control system. This system, completed in 1926, was designed to handle aircraft having a maximum speed of 95 knots.

The computation in these early analog computers was performed entirely with mechanical cams, differentials, multipliers, component solvers, and integrators. With the exception of the electrical contact-type servos, the reliability of these mechanical analog computers was controlled almost entirely by the mechanical designer and the people in the machine shop. Improvements in the performance of these computers were obtained over the years through a better mathematical understanding of the dynamic fire-control problem and more precision in the design and production of the various mechanical components. In service, the reliability of these mechanical computers was very good.

Just prior to World War II, a basic advance was made in analog computer technology—the introduction of the electrical-electronic computer. These computers used electrical components such as shaped potentiometers, electrical resolvers and synchros, and the servomechanisms were electrically driven with vacuum-tube amplifiers. This new concept resulted in the more rapid solution of the fire-control problem and some reduction in size, weight, and manufacturing cost. Unfortunately, these computers were much less reliable than their mechanical predecessors, primarily because of the poor reliability of the amplifiers. The reasons for this are clear now, although they were not at that time. The problem was twofold. First, the amplifiers were designed by engineers with little background of experience in the design of electron-tube devices and, second, the pressure of war and the rapid changes in requirements did not permit redesign to improve reliability before attempting production. Some of these computers, extremely promising in concept and basic performance, never reached service use because their electronic amplifiers were so unreliable. I might add that even more developmental failures in fire-control computers occurred during World War II because of a reverse situation in which experienced electronics companies tried to design fire-control systems without the necessary background in the basic fire-control problem. The lessons learned were very expensive, but they helped to establish one of the fundamental principles of the modern reliability concept. We know now that to develop a satisfactory and reliable military device requires a thorough understanding of the operational area involved as well as experience in the design techniques employed.

After the basic electrical analog principles were first developed, improvements in analog computers for weapon control came about largely through improved reliability, reduced size, and increased precision of the computing components and, most significantly, as a result of a more sophisticated and scientific understanding and treatment of servomechanism design.

World War II and its forced-draft research and development effort, together with the development of fire-control radar and more advanced weapons, pushed computer development forward rapidly. Before the war was over, the control of guns, aircraft, bombs, torpedoes, mines, rockets, and even guided missiles was being accomplished with the aid of analog computers.

Near the close of the war, a most significant weapon control concept was developed—the integrated fire-control system. Prior to this development, it was the practice for military agencies to build up a fire-control system from various pieces procured separately from different companies. As the speed and maneuverability of targets increased, with a corresponding increase in the performance and complexity of a weapon control system, it became necessary to develop the entire system under one system engineering management. The integrated weapon control system, now a more or less uniformly accepted concept, resulted in improved performance and substantial savings in size and weight. This principle of integrated system design must be given careful consideration in all future weapon control developments.

Between World War II and the beginning of the Korean conflict in 1950, the Military Services embarked upon a new era of weapon development generally based upon the kind of war that might be fought in 1960. Development programs which offered only marginal improvement in performance over World War II devices were discontinued, and emphasis in air defense was placed on weapons capable of engaging targets of near-sonic or supersonic velocity in mass saturation attacks. Guns gave way to guided missiles; manual control of interceptor aircraft was considered obsolete and the lethality of nuclear weapons was multiplied many times over.

Requirements for computers for the dynamic control of these new warfare concepts advanced rapidly, and a new kind of computer emerged, one which had the functions of keeping track of a multiplicity of targets, evaluating their threat to certain defended areas, assigning defensive weapons to individual targets and, in some cases, controlling the weapons themselves. The successful instrumentation of a computer to perform this complex of operational functions indicated the desirability—if not the necessity—of going to digital techniques.

This was the beginning of the era of "push-button warfare," and with it began a rapid transition in engineering thinking from analog to digital computers for weapon control. There was a lot of opposition to this on the part of many knowledgeable people in the weapon control field, both in the military and outside, most strongly pressed by those involved in airborne weapon control. It was argued that a digital computer of the size and complexity of the then current general-purpose machines could not possibly be condensed into a size and weight that could go into any aircraft. Furthermore, it was argued, even if by some miracle of engineering it could be so compressed, such a machine would contain so many vacuum tubes and other electronic components that it would be completely unreliable in service. (I might add that some of these thoughts are still prevalent among military people.) However, with the promise of more reliable computer components, such as semiconductors and magnetic devices, this opposition gradually softened and a few visionary people throughout the military departments initiated experimental developments of weapon control systems around digital techniques.

Looking at the weapon control picture today, I believe that the change to digital computing techniques is desirable and inevitable. In view of the rapidly increasing complexity of weapons of all kinds, I am convinced that digital methods offer the greatest promise for solving the control problems. Furthermore, the state of the electronic component art justifies the development of digital devices for all new weapon control programs. I believe that, in the future, analog weapon control will play a minor role in the support of digital systems.

I doubt that it is fully appreciated in the weapon control field that the digital computer promises many advantages over the analog device in addition to its greater performance capabilities. By the very nature of its instrumentation, the digital computer has far greater flexibility than an analog device; as a result, a single basic computer design, with only minor modifications, can be applied to the solution of a number of different weapon control problems. This capability has very significant implications with regard to standardization of design, which would result in economy of engineering effort, improved reliability, and enhanced production and logistic posture.

Another advantage that is of some significance in these times of steadily increasing cost of national defense is the fact that a digital computer is considerably cheaper to manufacture and will require less skilled labor. Also, the lead time to get a newly developed digital computer into production should be much less than for an analog device.

To substantiate these advantages, I have some comparative information on an airborne digital computer which is now entering pilot production as a direct replacement for an analog computer in an existing bombing-navigation system. It is estimated that the quantity production cost of this digital computer will be about 40 to 50 per cent less than that of the analog computer it replaces. Capital equipment required for production of the digital computer is expected to be reduced by 70 per cent; the requirement for skilled manufacturing labor should be reduced by almost 70 per cent, and the lead time for new production is expected to be reduced by 60 to 70 per cent.

These many potential improvements in the digital weapon control computer are very attractive. But there is a matter of major concern to many military people and systems engineers, which could seriously delay the widespread application of digital computers in weapon systems; that is the fear that system reliability may be seriously decreased. The reliability of electronic devices has not acquired a good reputation among military people, and they know that digital computers are electronic equipments.

I also share this concern, not because the reliability of digital computers cannot be made as good as, or better than, the best analog device now in service, but because, in entering this new field of digital technology, we may not fully use the knowledge of weapon control systems engineering and equipment reliability which has been developing in the electronics and weapons system industry.

The relatively new field of digital computers has been built up primarily around the requirements of the general-purpose machine. As in any new and highly specialized branch of engineering, there is a tendency here that a tightly bound group of specialists may develop, speaking its own language and tending to some extent to break away from other branches of the electronics industry. This has the effect of decreasing the interchange of technical experience—a potentially serious deterrent to both the reliability and systems performance of digital computers in weapon control systems.

As weapon and target capabilities have increased, the basic weapon control problem has changed little. The problem has become more complex and the requirements for solution more exacting, but the fundamental principles are the same. The only thing we are doing differently with digital techniques is to solve an old problem with new mechanization. We can waste a lot of time and engineering resources in this inevitable transition from analog to digital computing techniques if we do not make maximum and continued use of the weapon control know-how that has been built up in this country over the past quarter of a century.

We can suffer even greater losses if the proven reliability concepts and techniques established through years of hard work and cooperative effort on the part of industry and the military departments are not applied to the fullest extent in the military digital-computer field. After all, to obtain reliability, the techniques which must be applied in design, test, manufacture, operation, and maintenance are no different for a digital computer than for any other military electronic device of comparable complexity. Unquestionably, such methods as self-checking, which can be applied so readily to digital computers, will greatly assist in service maintenance, but they will not improve the operational reliability of a weapon system such as a guided missile or a high-performance interceptor aircraft.

With present techniques and components, I am convinced that we can design digital weapon system computers which will be more reliable than the best electronic equipment now in service. In a progress report on reliability of military electronic equipment, given before the Third National Symposium on Reliability and Quality Control on January 14, 1957, I used data on a digital bombing computer as an example of reliability improvement made over the past year. This kind of reliability can be achieved, however, only when the basic design of a device is thoroughly engineered for reliability and adequately tested before production is initiated.

Many times in the past two years I have discussed the basic steps in design, testing, production, procurement, maintenance, and use that are required to obtain a highly reliable military electronic device. I need not repeat these in detail here since they have been published widely in the technical press. But I do want to emphasize that the reliability of any electronic equipment is critically dependent upon the design engineer. If computer designers do not take into proper account the engineering principles controlling reliability, which are now well known, designs will very likely be unreliable in service, regardless of how sophisticated the logic may be and in spite of anything that can be done in the production line or by maintenance. Reliability can be controlled in manufacture and it can be maintained in service, but it can be established only by sound basic engineering in design.

One of the most promising techniques for obtaining reliability in digital computers appears to be the exploitation of their basic inherent flexibility to develop standardized designs of system building blocks. The basic geometry of many weapon control problems is quite similar and can be solved by proper system grouping of similar computer elements. Such a standardized design would make it unnecessary to develop a completely original computer for every new weapon system project and would permit the use of standard computer elements of proven reliability—reliability which could be brought to a very high level through extensive engineering, testing, reengineering, and continued production.

It may be argued that such a philosophy would seriously impede the advancement of digital computer technology. I do not agree. The real advance of digital computers in the weapon control field is going to result from more sophisticated weapon system engineering, advances in logic and improved component parts, not from a continued redesign of circuits and packaging.

At any given time, the same component parts are available to all computer designers—or, at least, they should be. Once circuits and packaging techniques, developed around these components to perform a particular computer function, have demonstrated a high degree of reliability, these circuits and packaging designs should be standardized and used in all applications to weapon control computers where an unacceptable compromise of weapon system performance would not result. Obviously, as new and improved components or techniques become available, new standardized designs should be developed around them. These designs, when proved to be better than those already in existence, should be adopted immediately.

In summarizing the advantages that can accrue to the military users from a design standardization program (some of which I have already mentioned), these factors are significant. The amount of engineering effort, cost and time required to develop a new weapon system would be substantially lessened. Also, the cost of production could be reduced because larger quantities of similar items could be manufactured, thus permitting the utilization of more economical manufacturing processes such as automatic assembly. Furthermore, the lead time required to get a newly designed weapon control system into production would be shorter. Another advantage to be gained from such a standardization program would, of course, consist of improvements in logistics, supply, and service maintenance.

I urge that those who are engaged in the development of digital computers for military weapons systems give careful consideration to this challenging problem of establishing and maintaining design standardization in this field. I can assure you that my office will make every effort to assist in bringing such a standardization philosophy into being as early as possible.

Another important need in connection with reliability in weapon systems employing digital computers is for increased emphasis on systems engineering. At present, digital computers are being developed to work in weapons systems in which other major system components were designed to function with analog computers. The input and output elements of these systems are analog and must be converted to operate with a digital computer. These conversions are costly in equipment complexity and they penalize over-all system reliability. Much more emphasis is needed on the development of various weapon system elements specifically designed to operate in a digital environment so that these costly conversions will not be necessary.

The last technique for reliability that I will present is simplicity. This, again, is a reliability axiom which is not unique to the digital computer field—but I suspect that it may be more difficult to achieve in this field than in other areas of military electronics. By careful design of logic and programming, much can be done to simplify the computer instrumentation in a weapon control system. We must have very careful systems engineering to make certain that we have the simplest system possible and that some of the solutions in the over-all weapon control problem cannot be obtained satisfactorily with less complexity and more reliability by using analog techniques.

In summarizing I would like to present these pertinent conclusions.

1) Because computers are vital to the operation of every modern weapon and weapons system, an extremely high level of operational reliability in these devices is absolutely mandatory.

2) The trend in weapon control is definitely toward the digital computer, because of its greater flexibility and higher accuracy and its advantages of lower cost, better producibility, shorter lead time, and lower requirements for skilled manufacturing labor.

3) The cooperative effort of the military departments and industry must be directed toward the immediate goal of standardizing the design of digital computer functional building blocks for application to weapon systems.

4) The successful use of digital techniques in weapon control will depend to a large extent upon the applica-

tion of combined experience in weapon control and digital technology.

5) The techniques for obtaining reliability in a digital computer are fundamentally the same as for any other electronic equipment of similar complexity. The principles for obtaining reliability of military electronics equipment through sound design, testing, and production controls are now fairly well established and should be applied to the fullest extent in new computer designs.

6) Careful attention should be given to systems engineering in the development of a weapon system employing digital computers to ensure that all system components are designed so as to minimize conversion of information between analog and digital forms.

7) Careful consideration should be given to logical design to obtain optimum simplicity of equipment design. Analog techniques should be employed for mechanizing functions where they are best for the purpose.

In closing, I would like to emphasize that they who are working in this relatively new field of digital computers have a great obligation in the defense of the country.

Many of the computing devices which are being designed are absolutely essential to military weapons and weapons systems, and they will become progressively more important as the capability and complexity of these systems continue to advance.

Although the challenge of making these new devices sufficiently reliable to be acceptable for military applications is great, there is a substantial background of knowledge and experience in reliability engineering to draw upon.

I see no reason why these new devices should not be completely reliable as they first become available to the using military services. If they are not, the future of digital computers for the dynamic control of weapons may be seriously affected.

Computers with European Accents

ARTHUR L. SAMUEL†

AS THIS is a luncheon talk, it should contain some humor but there is really nothing very funny about some of the European computer developments which are offering competition to certain unnamed American firms that are trying to peddle their wares in Europe. One of these competing computers, known as the GAMMA 3, is manufactured in France by an organization known as Compagnie des Machines Bull. Compagnie Bull has some 350 of the GAMMA 3 machines in the field. It is primarily a plugboard machine with 64 single-address instructions and can be compared in a general way with the IBM 604, although, more strictly speaking, it occupies a position intermediate between the 604 and the 650, particularly when an 8000-word drum extension unt is attached. The interesting features of this machine are not, however, the size, speed, or relative cost, which after all are quite comparable with American developments, but rather the extensive use of techniques which have never found wide acceptance in the United States. This refers particularly to the use of electromagnetic delay lines as storage elements, and a number of other techniques, the use of which has enabled this moderately small organization to compete with organizations many times its size. This is a virility which belies the all-too-prevalent impression of French decadence.

† Internat'l Business Machines Corp., Poughkeepsie, N. Y.

The same company has recently announced a complete data processing system called the GAMMA 60 which includes a central processing unit with magnetic corestorage. The peripheral equipment includes magnetic drums, magnetic tape units, both card and paper tape readers and punches, lined printers, etc., all under internal stored program control.

Professor F. C. Williams of Manchester University has made many contributions to the computing art, perhaps the most well-known being the cathode-ray storage system to which his name is customarily attached. He has gathered around him at the University a small group of very competent men who have made and are continuing to make substantial contributions. The main location of the Ferranti Company happens to be in Manchester, and, as one might expect, a cooperative arrangement has developed in which Ferranti contributes to the support of a computer project at the University. It profits, in turn, by the developments made there, and manufactures commercial computers embodying some of the University's developments. Several machines of a first design, known as the MARK I, have been made and are in operation at such diverse places as Toronto, Canada, and Rome, Italy. This computer was followed by the MARK I STAR, and more recently the Ferranti Company has announced a new large-scale computer known as the Ferranti MERCURY Computer. This is a high-speed computer, using floating point, with a

1024-word core memory, a 16,000-word drum, and 7 index registers and is quite comparable with the larger machines made in this country. A weakness of this, as well as of nearly all other European computers, is its dependence on punched paper tape as the primary input and output medium. This observation may, however, be biased because no one in the United States appears to have produced paper-tape equipment equivalent to that manufactured by the Ferranti Company. Several MERCURY machines are being constructed at the present time. One of the first machines is to go to Manchester University. A second is to go to the Norwegian Research Institute for Defense; a third will be installed at a Computing Center now being planned for the University of London, while Oxford University is getting a fourth. Mr. Brian Pollard, who is in charge of this activity at Ferranti, tells me that they have orders for 17. Altogether, there are some ten different industrial concerns in Great Britain making computers and they are reported to have orders for over 84 large computers on their books at the present time.

During the same symposium, Mr. Bill Elliott covered up a similar display of the letter "F" on the Ferranti PEGASUS Computer, or FPC, by saying that it stood for "Fast." Incidentally, the letter "P" originally stood for "Package." This was later changed to PEGASUS when the Ferranti Company waxed poetic and decided to name all of their computers after stellar constellations.

The Ferranti computer FPC I (to differentiate it from the FPC 3, a commercial version) is an amazingly fast computer in terms of its ability to get work done, although it is basically a small, fairly low-speed machine. These computers are currently being produced; 30 are on order, 2 have been delivered to customers, and one has been installed in a company-operated Computing Center at 21 Portland Place in London.

Most of the computers of Europe are binary rather than decimal. For example, the Swedish Board for Computing Machinery, after first building a relay computer called the BARK, later designed and built an electronic machine called the BESK. As originally built, the BESK was a 40-bit, parallel, asynchronous computer using Williams tube storage; in concept, very much like the Princeton machine.

However, here the resemblance ends. The construction details, the exact circuitry, and all the many different features which give a machine its character were distinctly original. Some of the more original features of this machine are the use of a dielectric paper tape reader which operates at 400 characters a second, and an unusual record for economy in the use of vacuum tubes to achieve the desired results with, of course, an astounding record for reliability. They quote figures like 85 per cent good time on a three-shift basis.

For years the Swedish Board for Computing Machinery has been living on year-by-year appropriations, not unlike the situation confronting certain government-supported activities in this country. Possibly for this reason the situation became critical roughly a year ago and almost the entire engineering staff left in a body and joined an industrial organization known as Åtvidabergs Industries. Dr. Havermark tells me that the present staff consists of 35 members, these being 13 mathematicians, 11 engineers for running and maintenance, 3 keypunch operators, and 8 employees for general administration. This group at Åtvidabergs is now busily engaged in building a copy of the BESK to be called the FACIT which will form the nucleus of a second computer center in Sweden. This machine is an exact copy and, consequently, incorporates all of the improvements which have been made to the original BESK in the last three years, such as the use of a 1024-word magnetic core memory. In addition to this work at Åtvidabergs, the BESK is being copied elsewhere in Sweden and in several different places in Europe. The Svenska Aeroplan AB, known as the SAAB, had, prior to the trouble at the Board, arranged to build a copy for their own use to be called SARA. Another, to be known as the SMIL is under construction at the University of Lund, although this is a stripped down version without core storage.

The Danish Academy for Technical Science is planning a copy of the BESK for their Institute of Computing Machinery, which is to be called the DASK. The Board for Mathematical Machines of the Royal Norwegian Council for Scientific and Industrial Research has also been considering a BESK to supplement the small magnetic drum computer called the NUSSE which was completed in 1953. However, the most recent information seems to indicate that the Norwegian Defense Research Institute is purchasing a Ferranti MERCURY and this may obviate the need for a BESK.

Over-all developments in Europe are following an amazingly similar course to that pursued in the United States, with some striking differences in timing. Many of the earlier machines were built by schools, others by government laboratories—perhaps rather more in proportion than here—and very few by industry. These machines were all plagued by difficulties of completion similar to those experienced in the United States. Recently, industrial concerns have been entering the field in Europe so that there are appearing a number, or are shortly to appear a surprisingly large number, of different machines, some of which are decimal.

Elliott Brothers in Great Britain had early exploited the possibilities of using nickel delay lines for storage and had built a computer known as the NICHOLAS using these lines. As a result of this work, this company was commissioned by the NRDC (the National Research Development Corporation) to build a small computer. This computer, known as the 401, was unique at the time, for its use of a limited number of differently designed package units. After making three copies of the 401, the Elliott organization has gone ahead with a more pretentious design for commercial applications which is now being marketed as the 405 series of machines and for which there are said to be a dozen orders.

Having successfully launched the PEGASUS and the 405, the NRDC is now turning its attention to commercial data processing assemblies, with the word "assembly" used advisedly.

It has contracted elsewhere for the design of an all-transistor-driven core logic and core-store data processing assembly in which the main feature will be a marshalling yard for information external to the computer. The strategic object of this will be to provide a device to which a number of independent keyboard operators can send information in an uncorrelated fashion. This is still in an early stage.

The main preoccupation of the Manchester University group at present is with the input-output facilities of the computer which, as I have indicated, follow European rather than American practice. For example, the Manchester Group has found that a fair proportion of the results printed by their MARK I computer had to be subsequently plotted. They are, therefore, building a cathode-ray plotter which is 80 per cent completed. This plotter uses a 9-inch tube for visual observation and a second tube to be photographed by an automatic camera. Each coordinate of the beam is specified by the least significant 8 digits of a 10-bit word, thus providing a 256×256 array of dots which may be used.

Not content with this unit only, they are also building a roughly 10-inch square electroluminescent matrix panel which will plot an array of 512×512 points using the power law voltage characteristic of the phosphor to provide the discrimination. They hope to use direct contact photography for recording.

For high-speed numerical output, they are building magnetic tape units which operate at a maximum rate of 1000 characters a second, each character consisting of five binary digits. The magnetic head actually consists of two heads, one for writing and one for reading, separated by 20 mils, the read head being of the static reading variety. Characters are recorded on the tape at a fixed packing density of 50 per inch independent of the tape speed.

Using this same magnetic reading head, they are also constructing a tape editing unit entirely transistorized, with an input power of less than 10 watts, which can be used at teleprinter speeds of approximately 6.7 characters per second.

Turning to storage devices, the Manchester group has obtained a magnetic tape drive unit built by the Pye Company which drives the tape in either direction at a maximum speed of 100 inches a second. They intend to use addressed records, each containing 1280 digits, with the records sequentially addressed and provisions being made to exclude automatically imperfect regions of the tape.

This group is also turning to evaporated ferromagnetic films as a storage medium and they have built an evaporation unit which has all necessary facilities for rotating large substrates at elevated temperatures, etc.

The Manchester group attaches a great deal of importance to its autocoding system which they expect will virtually replace direct coding at a maximum expense for the worst possible case of a factor of 2 in computational speed. They have written a translation program which they liken to the IBM FORTRAN system and they are attempting a general program which will automatically solve any linear second order partial differential equation by finite difference techniques. All in all, this is quite an ambitious program for a small group at a University, but this is the way things are done in England.

Professor Wilkes at Cambridge University built the first modern stored-program computer in England, known as the EDSAC I. This computer is still in operation after many years of useful service, but its days are now numbered, since Professor Wilkes and his able associates are in the midst of building a second computer, the EDSAC II. In fact, that portion of EDSAC II which has been completed has been linked up with a temporary decoder using an abbreviated order code, and this is actually operating. Since this is a portion of the II machine, and since like all machines only 80 per cent completed, Professor Wilkes, in the true British tradition of understatement, calls this machine EDSAC 1.5. Professor Wilkes' group has programmed and has in operation an interpretative routine which will accept program codes for EDSAC I and will execute them faster than they can be run on EDSAC I. A problem in stellar structure is in process on EDSAC 1.5 which is similar to the problem that Hoyle and Haselgrove have just been doing on a 704 in Pasadena.

EDSAC II uses the microprogramming technique in its decoder which came out of the Cambridge work and has been described in the literature. The computer employs a rather unique packaging arrangement in which all of the components for each stage of the arithmetic unit are contained in one pluggable unit. Forty of these units are used to make up the 40-bit accumulator and a substantial number of the same units are used in various other parts of the machine. In terms of order complexity and speed, this machine compares favorably with the better commercial machines. The EDSAC 1.5 will remain in its present form for two or three months when they will begin to install the EDSAC II control matrix.

Professor Biermann of Gottingen is an astrophysicist, and he and his able technical leader, Dr. Heinz Billing, at the Max Planck Institute, have built a series of computers known as the G1, the G1-A, the G2, and now the G3. The G1 is a small drum machine and has piled up an impressive record of 28,000 hours of operation with 82.3 per cent of this as useful time. The G1-A, a modernized version of this machine, is ready for its trial runs. This machine is controlled by photoelectrically read paper tape. Their second machine, the G2, has also been in operation for some time, although it is at the moment down for a general overhaul. However, the G3 is currently of the greatest interest. This is a parallel 40-bit binary machine with floating point arithmetic, designed, as were all of the Gottingen machines, for scientific computing. It will have a core memory for 4096 words and will have adequate indexing features for automatic address modifications. Incidentally, this machine uses a whole word for each instruction, thus reversing the previous practice almost universal in Europe up to now of following the Institute for Advanced Study's practice of confining each instruction to a half word. Wired microprogramming is to be extensively applied in this machine and they are, at the moment, entirely revising their projected order code in an attempt to make it especially efficient for the use of computer programs.

The Max Planck Institute Computer group in Gottingen is shortly to move to Munich. This will make Munich quite an important center as far as computers are concerned since one machine, the PERM, constructed at the Munich Institute of Technology, is already located in this city.

The PERM, a parallel magnetic drum machine, is a cooperative venture between the Electrical Engineering Department under Professor Piloty and the Mathematical Department under Professor Sauer, now Chancellor of the Institute. This is a fine example of a fast drum machine which, with some projected improvements, will become a very good machine indeed. Their drum, which runs at 15,000 rpm, is extremely quiet. At the moment they are just recovering from troubles with the contacts on their pluggable units, these being ordinary tube sockets which had to be replaced—quite a formidable task for such a small group.

One other group in Germany deserves special mention, this being the Institute for Practical Mathematics at Darmstadt under Professor Walther. Their machine, the DERA, a magnetic drum machine operating in floating decimal, is complete as far as construction is concerned and is now going through the final debugging stage. This group has also recently acquired a commercial machine of American design but manufactured in Germany. Work at these three places in Germany has been supported by the German government. A much larger number of universities are shortly to get computers of commercial manufacture. Some of these will be of foreign design, some even of foreign manufacture, but a substantial number of German firms are currently entering the computing field.

The firms of Siemens and Halske in Munich, and Standard Electric of Stuttgart (actually an affiliate of an American firm) are reported to be building transistorized computers. Two firms of the A.E.G. group, Olympia-Werke at Wilhelmhaven, and Telefunken at Backnang, are developing electronic computers. The firm of Zuse KG in Hunfeld, after a successful experience in producing relay machines, is now accepting orders for their Z-22, an electronic computer, about equal to its G1-A.

The ERMETH Computer which was designed at the Swiss Federal Institute of Technology is also a magnetic drum machine, decimal with floating point, which is currently running with a 400-word drum, although ultimately intended to operate with a 10,000-word drum which is currently not in operation because of magnetic head difficulties.

In Holland the Mathematical Center under Dr. van Wijngaarden and the P.T.T. have done work that is particularly worthy of mention. At the P.T.T. a group under Dr. van der Poel has designed a drum computer, called ZEBRA, based on the principles published by Dr. van der Poel. The S.T. & C. organization in England is building several of these machines; the first one should be completed during the next few months. Dr. van der Poel's ideas were quite novel when first proposed, and have been used in the Zuse machine.

There are some transistorized computers in Europe, such as the all-transistor machine built by E. H. Cooke-Yarborough at Harwell in England. There are perhaps a half a dozen other places in Europe where transistor computers are in operation or in an advanced stage of construction. Most of these are rather small experimental machines, and almost without exception they are rather slow by American standards. Transistor production in Europe has lagged behind that in the United States. This is particularly true with respect to high-frequency units, and this lack of transistors has inhibited their extensive use.

There are many other machines that should be mentioned, for example, the work done in the government laboratories in Great Britain, such as the National Physical Laboratories. Their first attempt, known originally as the ACE, was used as a basic design for the machine now being manufactured by the English Electric Company as the DEUCE. Meanwhile, the N.P.L. is going ahead with a new ACE machine which will be several times as fast as the DEUCE.

Europe is perhaps behind the United States in computer developments and we need fear no immediate reversal in relative positions. However, there are many clever people in Europe; they have a tradition in England of achieving a lot with a little, in Germany of thoroughness, and in France of mathematical intuition, to name but three countries. These people are not going to permit us to continue in undisputed mastery of this expanding field. We can expect many new ideas to come from Europe. European accents, this time in computing, may again be heard in this country. European concerns, particularly those in England and in Germany, are known to be looking with envious eyes to the American market and it may not be long before they are offering their wares at prices which will be highly competitive.

Reliability from a System Point of View

ALEXANDER W. BOLDYREFF†

THE DEVELOPMENT of complex electronic and electromechanical systems during the past fifteen years has been guided primarily by considerations of improved performance.

In this development, perhaps the most outstanding factor has been a systematic effort to minimize human error by a maximum utilization of automatic or semi-automatic devices.

While the progress in this direction has been truly remarkable, it has been achieved at the expense of ever increasing complexity and cost.

A few examples will illustrate this point:

1) Number of vacuum tubes on one destroyer:[1]

Year	Number of Tubes
1937	60
1944	850
1952	3200

2) A modern mobile search radar for ground defense is composed of:

500	vacuum tubes
2000	resistors
1500	capacitors
300	transformers

as a part of a complete itemization of more than 20,000 replacement parts.

3) The Norden bombsight of World Wár II could be carried by one man and cost 2500 dollars. The computing bombsights of today weigh between one and two thousand pounds and cost more than a quarter million dollars.

It is not surprising to find associated with this growth in complexity an alarming rate of failure of equipment, and an ever increasing requirement for inspection and maintenance.

Thus, during World War II more aircraft were lost due to deterioration than were lost in combat.[2] Again, quoting from World War II experience:[3]

1) Sixty per cent of the British radars shipped to the Far East were found defective on arrival. Of the remaining 40 per cent, arriving in operating condition, half deteriorated on the shelf.
2) For a set of U. S. bombsights, 60 per cent failed as a result of poor packaging and rough handling, 15 per cent failed due to improper maintenance and overhaul, another 10 per cent of the failures were attributed to poor design.

The situation is no better today, and acceptable performance standards for complex electronic equipment are possible only at the cost of extensive repair and maintenance facilities. For military electronics, an estimate of the maintenance bill is from ten to one hundred times the cost of original equipment. Considering the number and the caliber of technicians required to service adequately existing electronic equipment, it does not seem possible that, if the present trends continue, the training of technicians can keep in step with the demands for their services, particularly in the event of total mobilization.[4]

The seriousness of the reliability problem has been thoroughly recognized now for a number of years. A great deal of work has been done to acquire a better understanding of this problem. Various methods of improving reliability have been advocated during the past nine or ten years.

It has been pointed out[5] that in the case of aircraft, after nearly half a century of experience, suitable reliability was attained only through redundant design; so that in case of failure of one component, another could be substituted in its place. In this way, even though some kind of failure (requiring emergency service outside the normal maintenance routine) may occur in aircraft every seven and a half hours of flying, the ratio of failure to disaster is ten thousand to one. This ratio is one to one for the systems which are serial in nature, such that the failure of any one component leads to system failure, as, for example, in the case of guided missiles.

Considering the complexity of many systems in use today or in the process of development, it is not surprising that a great deal of emphasis has been placed on the importance of component reliability.[6–8]

† The RAND Corp., Santa Monica, Calif., and Univ. of Calif., Los Angeles, Calif.

[1] Prog. Rep. on "Reliability of Electronic Equipment," by the Ad Hoc Group on Reliability of Electronic Equipment for the Committee on Electronics of the Research and Development Board, EL 200/17, vol. 1 and 2; February 18, 1952.

[2] D. C. Kennard, discussion of paper by J. M. Frankland on "Criteria for Specifications," Res. and Dev. Board, Shock and Vibration Bull. No. 17, March, 1951.

[3] R. R. Carhart, "The General Problem of Reliability in Missile Systems," The RAND Corp., paper S-4; July 9, 1951.

[4] G. B. Devey, "Reliability in electronic equipment," PROC. IRE, vol. 38, pp. 344–345; April, 1950.

[5] L. N. Ridenour, "The Philosophy of Guided Missile Design," Res. and Dev. Board, Shock and Vibration Bull. No. 18; August, 1951.

[6] R. Lusser, "A Study of Methods for Achieving Reliability of Guided Missiles," USNAMTC Tech. Rep. 75; July 10, 1950.

[7] R. Lusser, "General Specifications for the Safety Margins Required for Guided Missile Components," USNAMTC Tech. Rep. 84; July 10, 1951.

[8] R. Lusser, "The Statistical Aspects of Reliability," *Electronic Applications Reliability Rev.* RETMA, no. 2; 1953.

Particularly noteworthy in this connection are the ARINC Study, the Signal Corps-Cornell University Program, the Vitro Study, the Bell Laboratories Studies, the RETMA, the JETEC, the AGREE, etc.[9]

At the same time, a great deal of effort has been and is being expended on component testing and inspection before they are employed in complex systems. But while all this is highly necessary, it may be far from sufficient to insure sufficiently high reliability of a complex high-performance system.

Let us define reliability as the probability of failure-free operation, for a specified length of time, in a specified environment.

For a serial system of n components, such that the failure of any one component causes system failure, it is possible to estimate the system reliability in terms of the (geometric) mean component reliability.

Consider a system of 500 components. For systems with reliabilities of 0.70 and 0.95, the mean component reliabilities are 0.99929 and 0.99995, respectively. Thus, it may be argued that the reliability of a system can be increased from 0.70 to 0.95 by an improvement in mean component reliability of only 0.07 per cent. But, of course, this reasoning is misleading. Component improvement means decreasing the probability of failure. In the example under discussion, to improve system reliability from 0.70 to 0.95, we would have to decrease the probability of component failure from 0.00071 to 0.00005, and this means that we must eliminate more than 90 per cent of failures for components which are already highly reliable. To do this for each of the very many different components of many complex systems now in the process of design is patently impossible, even at a prohibitive cost in time and money.

Let us return to the definition of reliability. To be operationally significant, this definition must be quantitative; *i.e.*, the reliability of various components, or systems, must be represented by a number. For vacuum tubes, this is frequently expressed in terms of mean life to failure. Unfortunately, this quantity is not a characteristic constant. Thus, a vacuum tube may have a mean life of 10,000 hours in ground equipment, 2500 hours in aircraft, and 13 minutes in a missile.

It is, therefore, impossible to speak of the component reliability without specifying the particular system in which it is employed, as well as the way in which the system is going to be used. And this includes the handling, packing, transportation, and storage, as well as the operational use. Certainly the rest of the system constitutes an important, sometimes the most important, part of the environment in which a given component must operate.

This brings us to the question of compatibility of various components and subsystems. To use an example, are the electromechanical, electronic, and optical components of the guidance system in a guided missile compatible with the ram jets or rocket engines of the propulsion system?

[9] "Reliability Factors for Ground Electronic Equipment," ed. by Keith Henney, McGraw-Hill Book Co., Inc., New York, N. Y.; 1956.

Such questions cannot be answered except by a comprehensive systems approach; I believe that the problem of reliability cannot be solved satisfactorily without the use of systems approach, and this means both system analysis and system synthesis.

Reliability, after all, is merely one of the parameters of a given system, the other parameters being performance, complexity, cost, logistic requirements, etc. These parameters are interdependent. Thus, improved performance generally implies greater complexity, lower reliability, and higher cost.

Let us consider the various steps in a system development program. These are five in number:

1) Definition of system objectives in terms of performance requirements.
2) Research, or investigation of alternate reasonable means of achieving system objectives.
3) Development, or selection and perfection of the best means.
4) Prototype test, or verification of performance, and determination of causes of failures.
5) Design and production, or final choice and manufacture of well engineered, reliable systems, capable of reliably meeting performance requirements.

Note that the degree of success in any step depends on the preceding steps.

Thus, the choice of performance requirements will usually dictate the complexity of the system and the tolerances of all the components. Yet this is often done in the absence of sufficient factual data or rational analysis and most often done by administrators or executives, who technically are the least competent to make these decisions. It is my opinion that here is the greatest source of low reliability, dooming many projects to ultimate failure. Certainly the most crucial question is whether the required performance is either actually necessary, or even technically attainable at a reasonable cost and in a reasonable length of time. Frequently, this question cannot be accurately answered without considerable research, development, and test, the feedback from which should dictate necessary changes, although in many cases substantial increase in reliability can be bought by relaxing unnecessarily stringent performance requirements. However, it is very difficult to point this out to the project engineers. Altogether too many of them treat the initial system objectives and exact performance requirements as sacred, and insist on freezing component and system design before there is a chance to subject system objectives to a critical analysis or to obtain feedback from actual performance tests.

The last and the most important point is that engineering is an art that can be practiced successfully only on a firm base of scientific fact. Reliable design is impossible for an unknown environment.

How much effort is expended on basic research? Let us take as an example the annual budget of our Federal government: of the seventy billions, forty-five billions, or about sixty per cent, is spent on defense. Of this sum about two and a half billions are earmarked for research and development, and only six per cent of this figure is to be devoted to basic research.[10]

This was emphasized in a recent report to the Congress by the Commission on Organization of the Executive Branch of the Government: "Among the Federal Agencies devoted to research and development there is but a minor amount of basic research into the laws of nature and the nature of materials. Yet the safety, the increase of productivity and the advancement of health in our Nation must come from constantly increasing knowledge through fundamental research. From these explorations come knowledge, discoveries, invention, and progress."[11]

[10] "Federal Funds for Science" (The Federal Research and Development Budget, Fiscal Years 1953, 1954, and 1955), Natl. Science Found., U. S. Government Printing Office, Washington, D. C.

[11] "Research and Development in the Government," a report to the Congress by the Commission on Organization of the Executive Branch of the Government; May, 1955.

Design of Experiments for Evaluating Reliability

JOHN HOFMANN†

The Nature of Experimentation

Introduction

THE consulting statistician is frequently charged with the analysis of large masses of data, and with drawing conclusions and making recommendations from the results. The data are usually collected according to the time-honored techniques peculiar to the field of endeavor represented. This often means that there have been collected, without control, observations on many variables. Some are related and others unrelated to the problem at hand.

The statistician has techniques for the analysis of such data, which sometimes are applicable after a few assumptions are made. However, often the only recourse is to curve fitting, or regression, *i.e.*, an attempt to fit a surface to the data taken, assigning one variable as dependent and the others as independent according to some rationalization. The problem in such analyses is usually the magnitude of the computations. Surface fitting with statistical methods usually involves matrix inversion, and it appears sometimes as though the matrix associated with any worthwhile undertaking is invariably large.

The design of experiments, with "design" used in the statistical sense, can be considered a means of reducing the computational problems by controlling the independent variables. However, before we pursue this point, it seems worthwhile to turn philosophic and to probe the meaning of the word "experiment."

Certainly, we are all familiar with experimentation. We do it every day without philosophical probing. Asked to define an "experiment," however, those who are trained in physics or chemistry will give a far different answer than will an engineer or a microbiologist. The meteorologists, astronomers, and biologists who must deal with available data may be completely lacking in ideas on the subject.

† Systems Labs. Corp., Sherman Oaks, Calif.

Webster defines an experiment as "a trial or special observation made to confirm or disprove something doubtful, especially one under conditions determined by the experimenter; an act or operation undertaken in order to discover some unknown principle or effect or to test, establish or illustrate some suggested or known truth." From this definition, certainly an acceptable one, we can note at least two kinds of experiment: the absolute experiment, exploratory in nature, planned to add to our fund of knowledge some new facts; and comparative experiments, designed to compare two or more theories, processes, or products and yield data on which to base an administrative decision.

The distinction seems, at first glance, to hold up. Millikan, measuring the charge on the electron, or Joule, measuring the mechanical equivalent of heat, are adding to our knowledge of nature. On the other hand, a production engineer, comparing the yield and precision of two machines, or an electronic engineer, comparing the output or service life of a black box to corresponding specifications, is seeking data, by means of a comparative experiment, on which he may base a decision—usually one with important economic consequences to him.

Actually, there are many in-between types of experiment. For example, the determination of atomic weights, apparently an absolute, knowledge-contributing experiment, is in reality a comparative experiment since it involves the determination of ratios of atomic weight. In fact, I suspect that most fundamental research is, in effect, a compromise between a desire to discover something new and significant, and a need to provide a basis or justification for making some administrative decision. Fortunately, the distinction is not too important since, in general, the statistical techniques, which are the subject of this paper, are similar.

The Scientific Method

While we are digressing, it would seem worthwhile to establish another idea at this point. Since we are scientists of one kind or another, either theoretical or applied, we all have some concept of the meaning of "scientific methodology." While it is true that there is room for debate as to whether there is *a* method, rather than many methods, there is sufficiently general agreement on some points to make a noncontroversial discussion possible.

For example, we can define the scientific method briefly as the application of logic and objectivity to the understanding of phenomena. The basis of a scientific method is the examination of what is known for the purpose of deriving therefrom theories, or hypotheses, which may be subjected to experimental verification. The oft-stated quotation, "Statistics can prove anything . . . " is quite false. We can derive proofs only by deductive logic in the manner of theoretical mathematics, and the proven theses are adequate descriptions of "nature" only to the extent that the assumptions and axioms on which they depend are also related to nature.

With a scientific method, then, we can only tend to verify and to add strength to our belief in an hypothesis. Therefore, we have in science a feedback system that is not unlike those with which designers and users of computers are very familiar. From past experience and observation, our own and those of others, we derive, by a logical deductive process, new theories or hypotheses. From the consequences of such theories we arrive at phenomena which should be observable if the hypothesis is true. We perform experiments to observe these phenomena and, from the results, our belief in the hypothesis is either strengthened or shaken. This information is fed back into the theory for the formulation of new hypotheses.

The experiment plays a central role here, in the sense that the results of the experiment to a large extent dictate the next step. What is frequently not recognized is that the experiment proves nothing. If the results correspond closely to those predicted by the theory, we are encouraged and our belief is strengthened. Otherwise, we are led to wonder about the validity of the theory.

We must recognize, of course, that much experimentation is done simply to explore. There is no hypothesis, only a desire for more knowledge about the phenomenon in question. The principles that we will discuss are still applicable, however. The difference is that we wish to estimate the parameters that describe the phenomenon rather than test some hypothesis about it. While this difference seems fundamental the basic considerations for the design of the experiment are, fortunately, much the same.

The Design of the Experiment

Preliminary Considerations

The one question that the statistician is most frequently asked is, "How large a sample must I take?" Usually, this question is not easily answered. Before any answer can be attempted many other questions must be answered, questions which are too seldom asked.

First, what is the purpose of the experiment? Although generalization is seldom wise, a great many experiments are conducted with no clear statement of purpose. Even the phrase in the title of this talk is too general to be useful. Evaluation of reliability is the subject of an example discussed here.

There is the lack of a commonly accepted definition of reliability. How can we establish the purpose of an experiment if we cannot agree on the definitions of the terms we use to describe it? Even assuming an acceptable definition, however special, there are other questions which must be answered. Our purpose may be to compare the reliability of a proposed new component to some previously established standard. In this case we may plan to test the hypothesis that the component reliability, r, is greater than or equal to the standard, r_0, with the alternative that r is less than r_0. Or, we may wish to measure the effect on reliability of variations in the environment in which the component will have to operate. Here we may have an estimation problem—that of estimating a curve or surface relating reliability to the severity of the environment; or we may wish to test the hypothesis that the effects of environment do not degrade reliability beyond a reasonable amount, *i.e.*, that the reliability in a severe environment is the same as that in a relatively mild one. Or we may wish simply to explore the relationship between reliability and some factors which may affect it, such as input voltages, minor design variations, etc.

It should be obvious that we must determine what can be measured in the experiment and relate these measurable quantities to the characteristic in question. For example, if we chose to define reliability as "the probability of satisfactory performance for the required length of time," then we must define satisfactory performance unambiguously in terms of observable characteristics of the test units. We must also relate these characteristics to time and to the system of which the component is a part. (As a simple example of the complexity of this problem, consider an electronic assembly whose output voltage is established as the sole criterion of performance. If we are monitoring this voltage and measuring operating time as the variable to be related to reliability, we frequently encounter this situation.

Performance specifications require that the voltage lie between fixed limits. A momentary transient drives the voltage outside these limits, but only for a short time. Shall the unit be considered to have failed when the output first went out of limits; or, since it returned within limits, shall we not regard this as failure? The answer, of course, can only come from consideration of the effects of this momentary lapse on the performance of the remainder of the system. This consideration may require another experiment to obtain the answers needed.)

As yet we have not even mentioned the considerations which are usually related to experimental design. None of the problems mentioned so far are statistical in nature. However, without the most careful planning and organization of thought on these preliminary problems, the remainder of the steps may hardly be worth the taking.

There is one further consideration that must be settled. What is the population that is being observed? Do we have a sample, or are we observing the whole population? This question is not so trivial as it sounds since to a large extent it determines the kind of inference that can be drawn from the observations. To make this clearer consider an example.

Suppose in a simple system a given voltage input should result in a certain measurable response—say an output voltage, and we are concerned with the effect of high ambient operating temperatures on the output. If there are four units to test, then we must first ask, "From what population can we consider these a sample?" (For reasons that will be more apparent later, it would be preferable to consider this a random sample.) "If they were made in a model shop and are prototypes of a proposed new system, can we validly extend any conclusions we may reach to include future units made on an assembly line basis?" This is another of the questions that the experimenter must answer. His conclusion should be based on careful thought, judgment, intuition and experience—on every bit of outside information available. It is not one to be arrived at lightly, as recent experiences with missiles have shown. The same sort of considerations are equally important in all other types of experimentation and they must be resolved by the subject matter specialist—the statistician can help him only to the extent of advising on the risks involved, and in the formulation of answers as to what the populations sampled may be.

In this example our test units may be a sample from one or more populations, but by observing at one or more temperatures we are creating several more—the populations of output voltages of similar units operated at similar temperatures. The problem is thus compounded by the fact that it is to these populations that our results pertain. In particular, we may establish the hypothesis that temperature has no effect on output, meaning of course no practical effect, within the temperature limits that we expect to encounter. We are saying, in effect, that for the population from which our units are a sample, the populations of output voltages corresponding to the temperatures at which tests are conducted are not different. To be a little more precise, we are saying that these populations have the same location (on the temperature scale) as measured by the average or some other statistic, and the same variability as measured by, for example, the rms error about the average.

To summarize briefly then, before the statistician can be of much assistance, the experimenter must state precisely his objectives, the hypotheses to be tested (or the quantities to be estimated), the variables to be observed and their relationship to the objectives, and the populations to which inferences are to be induced from the sample.

The Role of Statistics

When we plan an experiment, our purpose is to achieve one or both of these objectives:

1) To test an hypothesis concerning the magnitude of an effect,
2) To estimate the magnitude of an effect.

In the first case we have an hypothesis which we will accept or reject on the basis of our results. If we reject, presumably we decide in favor of some prestated alternative. We can arrive at the wrong conclusion in either of two ways. If the hypothesis is true, the inherent variability of our observations and sampling errors may lead us to reject it. Also, though the hypothesis is false we may be led to accept it. Nothing in the experiment itself can tell us if we are right or wrong. Thus we are concerned with making the probabilities of committing these errors as small as possible. These probabilities are functions of several aspects of the experiment that we can control, and of some that are out of our hands.

To the extent that generalizations are true, all other things being equal, the greater the sample size the smaller the chance for error. Most frequently, however, the sample size is controlled not by desires for minimum risks but by considerations of costs, manpower, available test equipment, and test units and allowed time.

A second determining factor is the inherent variability of the observed variables. In general, for the same sample size, all other things being equal, the less the variability the smaller the chances of error. The reasons for this will be considered shortly.

Finally, the design of the experiment, the assignment of values of the controlled test factors (treatments is the word usually employed here—a carryover from the fact that the statistical theory of design of experiments was developed, primarily, for agronomists who first recognized the need) can do much to reduce the chances of error for fixed available resources. This last consideration, an important one, leads to a specialized branch of statistics—the design of experiments—that is in itself too complex a mixture of art and science to discuss here in other than the most elementary terms.

In the example that we have been discussing we may have the hypothesis that temperature has no (practical) effect on output (within the range of temperature considered). We may conclude that no effect exists or that there is an effect. In either case we may be wrong. The probabilities associated with each error depend on the number of observations we take, the variability of the test units, and the experimental design. The most desirable situation is for the experimenter to specify the risks he is willing to take and then, with the statistician, to determine a design that provides those risks. More often than not, the only thing available to vary is the design of the experiment, since sample size is fixed by economic considerations and the material available for testing, and the variability of the observed quantities is inherent in the test units. In such cases it is important that the experimenter realize that there is risk of error and, where possible, try to obtain some measure of that risk.

If the purpose of the experiment is estimation, the design considerations are not much different. The probability that the estimate will be near to the unknown quantity estimated depends on sample size, variability, and in a less measurable way, on the design of the experiment. We must recognize, however, that the estimate is just that—it is a random variable and has a distribution (or is a sample of size one from some population) which depends on the population sampled and the size of the sample.

If we have given due consideration to the plan of the experiment we will have an estimate which will, on the average, be close to the unknown quantity estimated. A properly conducted experiment will also give us an estimate of the error of estimation, *i.e.*, the tendency of the estimate to deviate from the quantity estimated. This latter quantity is all too frequently ignored in the presentation of experimental results. Although it seems somewhat less than honest for the experimenter to present his results as "fact" without any statement of possible errors involved, the tendency to accept experimental results as "truth" is a persistent one, and the estimated error is frequently ignored or discarded.

There is an additional feature that the statistician should insist on in the design of the experiment. This feature is randomization—a word that is easy to use and hard to define. At some stage in the planning of the experiment, or preferably at several, a conscious effort should be made to introduce randomization. Without it the validity of the experiment is questionable regardless of other considerations.

Randomization may take several forms. For example, we should be able to regard our test units as being chosen at random from the population sampled. If the population is a real one—for example, the output of a production run—then a random sample is one which gave *every* member of the population the same chance of being in the sample. Otherwise we cannot be sure that the sample observed does not consist of items on which special care has been taken in their assembly (this has frequently happened). Thus to obtain a random sample we could assign a number to each member of the population and draw a corresponding sample of numbers "out of a hat" or from a table of random numbers.

If the population is a conceptual one—for example, all the observations of output voltage that might be made on one of our units—under given conditions we are on reasonably safe grounds in assuming randomness. But, if the unit is not a random sample from the population of such units, our observation is not a random sample from the population of all observations that might be made on all such units, and our induction must go from the sample to the population sampled at random.

One may well ask, what of the case where our test units are all that exist? Then we must define a conceptual population from which these units can be regarded as a random sample. Then, as already stated, in extending our induction beyond that population to some other, we are depending on our judgment in the field of application and not on any statistical considerations.

There is another form of randomization that is important. This is the random assignment of treatments or other experimental conditions to the observations. This is best explained by illustration. In our example of four test units we could consider two plans, among many. First we can observe all units at one temperature, then all at a second, and so on. In this case a random assignment of conditions would consist of random order for observing units at a given temperature and random order of temperatures. As before, randomness can be achieved by assigning numbers from a hat—with vigorous mixing before and between draws.

Alternatively, we might, if only two temperatures were involved, observe two units chosen at random at one temperature, assigned at random, and the other two at the second temperature. Both randomizations would be accomplished as above.

The purpose of this randomization is to assure that the results are independent of any intended or accidental effects due to purposive choice and, in any event, to assure the validity of any induction made from the results. Although the validity of the interpretation of the experiment depends in many ways on randomization, this feature of statistical design of experiments meets the greatest opposition. The reasons for the opposition, though taking many forms, can be reduced to one or more of these:

1) There is no reason to expect bias of any kind in the experiment,
2) It means a great deal of bother,
3) Things are more likely to get mixed up,
4) It is not necessary.

We can argue with equal validity that there is never reason not to expect bias. At any rate, if the experiment is worth doing at all, then it is worthwhile to take care and precautions adequate to assure its validity. Why waste time and money on a shoddy effort? Randomization gives the only assurance of valid interpretation of the experiment. If there is some unsuspected source of correlation between observations, randomization destroys the correlation mechanism and assures a valid statistical analysis based on the assumption of independent observations.

In our example, it may be that observations made on the same unit are time correlated. This frequency occurs with electronic devices. In the first design— all units observed at all temperatures—random order of applying treatments gives us assurance that if the average over units at one temperature differs from that at another temperature the difference is due either to chance variation or to temperature, but not to some other unknown effect. Random observation of units at the same temperature guarantees that differences between units occur solely due to chance and not to other unexplained factors.

In the second design—two units at each of two temperatures—if we assign the units without some mechanism of randomization, we cannot separate differences between units from different temperature effects by any statistical method or ingenuity.

The Analysis

The appropriate analysis of the results of the experiment is dictated primarily by the design. However, whether our purpose is to estimate the magnitude of an effect or to make a comparison (test an hypothesis) we will want an estimate of the effects and of the associated error of estimation.

If we want estimates it is not enough to give an average alone—a second sample of observations is very unlikely to give the same results. In fact, if there is no variation between results, there is no justification for multiple observations. Thus, if we admit that variation is present in the observations, we must admit to its presence in the average. The average is an estimate of an unknown quantity. An estimate of the variability allows us to construct an "interval" and quote "odds" that the interval includes the unknown quantity being estimated. These "confidence intervals" should be as narrow as possible, a feature which is achieved either by controlling error by appropriate use of the tricks of design of experiments, or by increasing the sample size.

If the purpose of the experiment is a comparison, we are intending to compare estimates of similar unknown quantities from two groups, or to compare a single estimate to some standard. In either case, variation is present. The statistician defines a significant difference in terms of a comparison of the observed average difference to the estimated variance (mean square error) of the difference. That is, to the statistician absolute difference is never significant. It is important to note, also, that a significant difference is not always important. A difference which is large compared to the variation represents an improbable occurrence if there is no real difference. It is called statistically significant. The degree of real difference that can be detected as significant with reasonably high probability depends on the magnitude of the variation. So here again, we improve our chances of a "correct" decision by control of errors (design of experiments), or by increasing sample sizes, or by a combination of the two.

Thus, whether our purpose is estimation or comparison, close attention to error control, *i.e.*, to the experimental design, yields dividends in terms of increased precision or increased chance of arriving at correct conclusions. It seems useful here to emphasize the distinction between the words "validity" and "correctness" as defined here. We are in a perpetual contest with nature. A valid decision regarding the state of nature is one which, whether correct or not, is properly arrived at as the best one possible from the data at hand.

Until further evidence is available, we can only attach a probability to the "correctness" of a decision. We can improve our chances, however, by the use of many of the available statistical tools. On the other hand, we can be certain of the validity of our results and consequent decisions by careful design and conduct of experiments which have built-in assurances that the assumptions essential to the planned statistical analysis of the data will be fulfilled. One such assurance, as already mentioned, is given by careful randomization.

An Example

At this point we could continue in either of two directions. We have mentioned the statistical theory of the design of experiments as a combination of art and science, and could discuss many of the designs, and associate analyses, that have proven useful. However, since many of the ideas presented here are not generally known, it seems most desirable to illustrate them with a simple example. It should be adequate to consider in more detail the illustration discussed previously.

First we will remove the restriction on sample size and plan for as many observations as needed. Also, we will restrict the study to the effect of temperature on output voltage. Suppose now that the unit to be tested represents a proposed new design. We want to estimate the effect of operation at high temperature and to compare this effect to operation at room temperature.

Since our units represent a new design, we will place a purchase order for the required number and these will represent the only ones in existence (at the time). However, although we recognize that we are sampling a (conceptual) population of units produced under similar conditions of relatively skilled model shop assembly, we are fairly certain that we can, with suitable allowance for increased variability of performance, extend our conclusions to the population of units that are mass produced. We might, for example, allow for a given increase in variability by increasing the stringency of our requirements on the prototypes.

Considering the availability of test equipment, manpower, and funds, as well as the time required to "soak" the units at temperature and take measurements, we decide on a total of about 60 observations. If we had a prior estimate of variability we could have considered, also, the risks of wrong decisions, but since this is a relatively complex procedure, let us assume instead that the experimenter will settle for a fixed probability of deciding that the design is inadequate, *i.e.*, that the temperature effect is too large to allow. Since a fairly large amount of money has gone into the design (a common consideration) and there is pressure to produce a usable unit (particularly common in industry and defense today) we establish that there is less than a 1 per cent chance of rejecting a satisfactory design (rejecting the hypothesis that temperature has no effect). When we have established the plan of the experiment, we shall be able to compute a curve giving the probability of rejecting the design (the hypothesis) as a function of the size of the real, but unknown, effect. If this curve is not satisfactory, we can modify the plan of the experiment.

Since, if we accept the design of the unit, we shall initiate production of it, we are concerned with variation between units. We shall, therefore, want to test several units at each temperature. Also since the unit, in use, will be operated repeatedly, the reproducibility of observation on the same unit in the same conditions should be estimated. This requirement calls for at least two observations per unit in each set of conditions under which it is operated.

Since reasonable judgment leads us to conclude that performance will be degraded by high temperatures, we decide that two temperatures, 70°–75°F and 180° ±2.5°F, will be satisfactory. If operation is satisfactory at both extremes, we can safely assume satisfactory operation at intermediate temperatures. This design will only allow linear interpolation to estimate operation at intermediate temperatures, but we do not expect to accept the unit unless operation is satisfactory at the extreme.

We have two hypotheses of concern:

1) Temperature has no effect on average performance (between 70° and 180°F),
2) Variability of performance is not dependent on temperature.

In addition we wish to estimate the variability in performance.

We are now ready to proceed to the design of the experiment. We have an upper limit of 60 observations. The units to be tested will, in normal use, be required to operate at any or all temperatures in the specified range, so we will want to operate the test units at both temperatures. We may, therefore, consider an experiment involving $4mn=60$ observations with m observations on each of $2n$ units at each of 2 temperatures. If we set $n=5$ and $m=3$, we have 3 observations on each unit at the upper temperature first and corresponding observations on the remaining 5 at the lower temperature first. Such a design allows us to obtain additional information as to whether high temperature has a lasting effect on operation. If we operated all units at the lower temperature first and then at the higher one, we would be dependent on subjective judgment for "proof" that any differences were due to temperature and not to some "temporal trend" in the observations.

Now a comparison of the average of the observations made at the upper temperature to that of observations at the lower temperature validly tests the hypothesis that there is no temperature effect on the average. A comparison of the average of the 30 initial operations to that of the second 30 checks for any time effect (a bit of "free" information not asked for), since each unit will be equally represented at each temperature, differences between units will not affect this comparison.

Finally, a comparison among low-temperature averages for the group operated at low temperature first with low-temperature average for the other group against the corresponding high-temperature averages will test for interaction between order of temperature application and temperature. Essentially, an interaction here means that the change in performance, if any, resulting from change in temperature is different when temperature is increased from what it is when temperature is decreased.

We require 10 test units (this could be reduced to 8 or 6 with resulting sacrifices in the protection against wrong conclusions) to be divided at random into two groups of 5. To accomplish this, we can draw 5 numbers from a hat, or, equivalently, from a table of random numbers and call this group I (low temperature first) or group II (high temperature first) as a result of a coin flip. These precautions are all that are needed to guarantee that differences between groups are due solely to the order of temperature in the group and that differences between temperature averages are due to temperature rather than to unconscious selection of units.

We will "soak" the 5 units in group II for sufficient time, we think, to stabilize the internal temperatures at 180°F and at the same time make observations on the group I units. If any initial warm-up of units is required before operation we should try to make this warm-up period uniform, but as an added precaution we can randomize the order of observing the 5 units in each group. (At the high temperature this will eliminate any effect due to inefficient "soak" time, which might cause the last units operated to have a higher internal temperature than the first ones.)

After the first operations are complete we will put the group I units in the oven, removing the group II, allow both to stabilize at the new temperature, and repeat the operation (and the randomization).

We must take account of the oven capacity in the design. It has been assumed that the oven available could accommodate 5 units. If this were not true appropriate modifications in the design would be necessary.

This experiment, with the built-in balances in observations and randomization to assure independence of observations and equal chance for uncontrolled effects (such as input voltage fluctuations) to affect any unit, is an example of a good experimental design for the stated purpose. Presumably, the size of the experiment has been limited by practical considerations rather than considerations of involved risks, this being the more common situation.

We have considered only one factor (temperature) but have taken precautions against another that might have caused trouble (time effect). We could, by further modifications, have arrived at a slightly more complex design to take into account any number of other factors (input voltage, for example). The principles involved for good design remain unchanged.

We can summarize this discussion simply by stating a few principles of good experimental design:

1) Try to achieve balance to simplify analysis; *i.e.*, try to have equal numbers of operations at each level of each factor,
2) Try to accommodate in the design all controllable factors that may conceivably be important, within the limits of available time and funds,
3) Randomize to eliminate any other effects that cannot be controlled,
4) If there is any reason to suspect that the effect of one factor depends on the level of another (an interaction exists between the factors) design to allow estimation of the interaction as well as the main effect,
5) Provide ample replication (repetition of observations under the same conditions) to obtain a good estimate of the error of estimation.

Conclusion

By way of conclusion I would like to quote from Prof. K. A. Brownlee of the University of Chicago a statement that I would wholeheartedly endorse.

"One overriding feeling I have is that often we try to do too much with too little. In our research work I often yearn for a little more care and craftsmanship, a little less haste, a little more thoroughness, a little more thought and a little less haphazard leaping around. The shoddy and inadequate experiment may often get us the right answer, as often by good luck as by good judgment, but I think that we would be better off in the long run if we did a more thorough job. Granted this may take a little more time, it is only because the work should have been started six months ago that we are in such a hurry today."[1]

Bibliography

The following books and articles represent a fairly complete but far from exhaustive bibliography of the statistical aspects of the design of experiments.

[1] Anscombe, F. J. "The Validity of Comparative Experiments," *Journal of the Royal Statistical Society A*, Vol. 111, Part 3 (1948), pp. 181–211.
[2] Bartlett, M. S. "The Use of Transformations," *Biometrics*, Vol. 3 (1947), pp. 39–52.
[3] Bose, R. C. "On the Application of Galois Fields to the Problem of the Construction of Hyper-Graeco-Latin Squares," *Sankhya*, Vol. 3 (November, 1938), pp. 323–338.
[4] Davies, H. M. "The Application of Variance Analysis to Some Problems of Petroleum Technology," *Journal of the Institute of Petroleum*, Vol. 32 (August, 1946), pp. 465–491.
[5] Eisenhart, C. "The Assumptions Underlying the Analysis of Variance," *Biometrics*, Vol. 3 (1947), pp. 1–21.
[6] Fisher, R. A. *Design of Experiments*, Edinburgh: Oliver and Boyd, 4th Edition, 1947.
[7] Frazier, D., Klingel, A. R., and Tupa, R. C. "Friction and Consumption Characteristics of Motor Oils," *Industrial and Engineering Chemistry*, Vol. 45 (October, 1953), pp. 2336–2342.
[8] Johnson, N. L. "Alternative Systems in the Analysis of Variance," *Biometrika*, Vol. 35 (June, 1948), pp. 80–87.
[9] Kempthorne, O. *The Design and Analysis of Experiments*, New York: John Wiley and Sons, 1952.
[10] Stevens, W. L. "The Completely Orthogonalized Square," *Annals of Eugenics*, Vol. 9 (January, 1939), pp. 82–93.
[11] Yates, F. "Incomplete Randomized Blocks," *Annals of Eugenics*, Vol. 7 (September, 1936), pp. 121–140.
[12] Youden, W. J. "Experimental Designs to Increase Accuracy of Greenhouse Studies," *Contributions of the Boyce Thompson Institute*, Vol. 11 (April–June, 1940), pp. 219–228.

[1] K. A. Brownlee, "The Principles of Experimental Design," Nav. Ord. Rep. 4028; June, 1955.

Reliability and the Computer

WILLIS H. WARE†

THE subject of reliability and the modern-day computer is far reaching and complex. The difficulties of designing and manufacturing computing devices which themselves exhibit long intervals of trouble-free operation are now fairly well understood, but many kinds of applications are yet to come in which a computer is but part of a larger system. For these, the consequences to the computer of a requirement for a reliable system may not yet be known. To some, reliability and a computer are mutually exclusive, but existing computing devices do operate trouble-free for long periods, and it is more than an accident that this is the case. This paper reviews a few salient aspects of the computer reliability problem, contributes a new viewpoint to some parts of the reliability question, and suggests areas where the techniques of the previous papers may be pertinent.[1,2]

There are many kinds of reliability: component reliability, which concerns itself with developing well-behaved building blocks; design reliability, which is concerned with the design technique and how, among other things, it can successfully combat environment, component drifts, and tolerances; manufacturing reliability, which is concerned with the fabrication process and how it deteriorates performance below design standards; and system reliability, which is concerned with the performance of the over-all system in operational use.

As a first attack on this discussion, it is appropriate to point out that in some respects the reliability problem for the computer is different than for other large classes of electronic gear. An example will be selected from the digital field, although an equally striking one might be found in the analog area.

It is customary in traditional engineering procedures—such as those for home entertainment devices or for some kinds of military equipment—to test the completed product exhaustively for operation under all conditions of its expected environment; and to demonstrate thereby its reliability. For instance, a radar set might be subjected to all combinations of temperature, humidity, supply voltage, supply frequency, and so on; and, so long as the set continued to operate and to locate targets within a prescribed tolerance, the design would be accepted as good. If the set then operated successfully and trouble-free for prescribed minimum intervals of time, it would be tentatively labeled as reliable. Finally, after it had demonstrated itself over extended operational periods with certain prescribed minimum maintenance requirements, it might be accepted as a reliable device. Notice, in passing, how different might be the meaning of reliability to the original designer, to the manufacturer, and to the operational or maintenance crews. Further notice that even if this radar set were required to demonstrate that it could locate all possible targets under all sets of environmental circumstances, it would still be possible to complete such tests within a reasonable time limit. The worst that might happen is that a large number of radial, circular, or spiral air patterns might have to be flown around the radar antenna while the set was subjected to various environmental or operational conditions. In any event, it is clear that a matter of months, or a few years at the outside, would be adequate to perform such exhaustive tests.

Consider in the same light the problem of reliability tests for a digital computer. It is especially appropriate for comparison, since the digital field has drawn many of its people from the older fields of electronics. For this example, assume a digital machine of very modest scale, say one with only a hundred toggles. If it has been designed as most machines of today, it will contain no redundancy, and hence, the hundred toggles are essentially independent variables. Each will contribute its own bit of information, and thus the number of possible internal configurations of the machine will be 2^{100}. One can visualize each of these internal configurations of the machine to be a vector in a 100-dimensional space; thus the life cycle of any particular problem or sequence of events within the machine will be represented by a path in this multidimensional computer space. If exactly the same approach to reliability testing is followed as was used in the radar example, each of these configurations will have to be checked. What does such a test program imply?

Suppose that one could test each configuration of the machine in 10 microseconds, which is a little fast even for today's achievements. There are of the order of 10^{13} microseconds per year, so that points in the 100-dimensional space could be traversed at the rate of 10^{12} per year. Now 2^{100} is roughly 10^{30}, so that, with no down time of the machine, the entire space could be covered in about 10^{18} years! The age of the earth is only approximately 10^{13} years. Furthermore, the test described would have checked the machine under only one combination of environmental conditions.

Certainly this is unreal and an extreme example. It does, however, have point. The number of degrees of freedom of the typical digital device is so large that con-

† The RAND Corp., Santa Monica, Cal.

[1] Alexander W. Boldyreff, "Reliability from a system point of view," this issue, pp. 18–20.

[2] John Hofmann, "Design of experiments for evaluating reliability," this issue, pp. 20–26.

ventional methods of inspection and of verifying correct operation are simply not applicable. Hence, design, inspection, and performance checks must unavoidably depend on statistical methods. Here is certainly one area where the techniques of experimental design should be useful. These methods should permit a fuller return for the amount of experimental effort expended.[2]

There is a further consequence of this inherent nature of digital devices. The designer must regard the computer as a statistical ensemble and admit that the parameters and characteristics of each kind of component will be statistically distributed. There results an extremely conservative attitude toward use of components in machines intended to give high performance. Because of this conservatism, meaningful performance statistics bearing on design faults are very slow to accumulate, and thus the feedback loop, which tells the designer whether his set of design criteria was good or poor, is extremely long. In one part at least of the computer field, reliability is different in these two ways: traditional techniques for evaluating it are not applicable, and statistics bearing on it are slow to accumulate.

This discussion leads naturally to a comment on an important difference between digital systems and some kinds of analog systems, insofar as the meaning of reliability is concerned. Consider, for the moment, how a mathematical analog machine of the differential analyzer type is constructed logically. The differential equation to be solved is expressed with the highest derivative on one side of the equality sign and with all else on the other side. The "all else" part of the equation is then mechanized on the assumption that the highest derivative already exists at some point. Finally, the equality sign is enforced by taking the output terminal of the "all else" mechanization and physically connecting it to the terminal on which the highest derivative had previously been assumed to exist. A particular kind of feedback has been forced upon the analog computer, namely, a logical or mathematical feedback which exists whether the computer consists of mechanical parts, electronic parts, hydrodynamic parts, or what not. Nonetheless, this kind of feedback behaves like a circuit feedback and may be positive or negative. Thus, if an errant electron chooses to travel the wrong way along a wire or to stop at the grid rather than at the plate of an electronic tube, in those cases where the logical feedback is negative, the malfunction will be masked and only a minor perturbation of the result will occur. The *form* of the solution is not changed, although the *precision* may be.

Every component of an analog computer is within at least one of these logical feedback paths. Hence, the analog machine, in many cases, is inherently tolerant of minor fluctuations in the performance of its components. Further, if the analog machine happens to be of the electronic variety, there are additional circuit feedback loops which encourage an amplifier to behave properly; it cannot misbehave if it wants to. Thus, in many situations, the mathematical analog computer need only have reliability on the average; instantaneously its components may indulge in a variety of deviations from normalcy and only minor consequences will result. Obviously, none of this is true when the feedback is positive or when a malfunction causes the feedback to change sign. Just as obviously, this argument also applies to analog systems other than differential analyzers, especially those analog systems in a closed-loop control application.

What is the corresponding situation for the digital system? If the computer is built as most of today's machines are—without redundancy—it must exhibit absolute or instantaneous reliability, because any component malfunction can cause difficulty. There are no logical feedback loops within the hardware of the computer to force the wayward component back into line. There may still exist circuit feedbacks in some parts of the machine—in video amplifiers for instance—but there are no equipment-wide logical feedback loops such as exist in the analog machine. There are, however, times when pseudo-feedback loops do exist by virtue of the particular routine in use; for instance, an error-reducing iterative process with automatic control of the error in the final answer will make the digital system tolerate some kinds of malfunctions. But, in general, such is not the case. In this respect, among others, the digital system is markedly different from the analog.

Evidently, in some respects, redundancy is to the digital machine what logical feedback loops are to the analog machine. Offhand, however, there does not appear to be, for redundancy, a parallel to the notions of positive vs negative feedback. It is interesting to speculate, however, on what deeper philosophical meaning is common to these two concepts, and on what use, if any, this might have in computer design.

As a result of this fundamental difference, the design of digital devices must be regarded in a statistical light. Components or whole assemblies of components must be regarded as having statistically distributed parameters and performance; the design technique must accomodate this fluctuation. As suggested before, the techniques of experimental design may be applicable. To date, most designers have concerned themselves not with the statistical distribution of the parameters of a component or circuit, but only with limit or end-of-life values of the parameters. Even so, experimental design may have much to contribute.

A further point comes from this discussion. The overall reliability or performance of a digital system may involve more than the reliability of the hardware alone. The particular routine in use may conceivably accommodate certain types of machine malfunctions, and in this respect, may better the apparent reliability. It is interesting to conjecture to what extent sophisticated programming of digital machines in this sense might increase their reliability.

The preceding discussion is not meant to imply that designers of analog equipment can be casual in their choice or use of components. While negative feedback activity can mask many troubles, it cannot, for instance, accommodate the catastrophic type of failure. However, it would seem that the digital design technique is more exacting than the analog design technique. While the analog designer tends to worry mostly about catastrophic failures and not to concern himself overly with long-term drifts and hardly at all with short-term drifts, the digital designer must concern himself with all of these and perhaps more. He must play the game with stricter rules and with more care than his analog colleague.

It does not necessarily follow that all analog devices are necessarily of high reliability. The analog art is a much older discipline than the digital art and has, therefore, tended to use the traditional techniques—for instance, exhaustive testing of a completed device to determine its reliability. The analog art also has tended to be used sooner than the more recent and relatively untried digital technique for the newer and more demanding applications. It was on hand and was used. Thus, before success finally came, there were the struggles and disappointments of the designers of airborne autopilots, navigational systems, and other analog computing systems. Further, at the time of the transition of the analog device from the laboratory to military and industrial applications, the need for an exacting design discipline was not fully realized, and many early efforts, for this reason, suffered serious setbacks.

The digital art has been fortunate in a way. Because the digital system inherently lacks the logical feedback paths, the problem of careful design for maximum reliability was evident from the outset. Consequently, a new kind of designer evolved who established new methods of design which could tolerate wide extremes of operating parameters. The whole of the electronic industry has profited in this respect from the growth of the digital field, but, unfortunately, the new wisdom and design techniques of these people are not being disseminated nearly widely enough nor rapidly enough.

Through the balance of the paper, the emphasis will be on the problems of the digital field. This is not an implication that the future of the analog device is limited or even doomed; it certainly is neither. The analog technique has successfully made the transition out of the laboratory and is a rather well-grounded, although perhaps not sufficiently documented, art. On the other hand, not much has been stated about the reliability problems of systems containing a digital device.

The digital art again has been fortunate in two ways. It was first applied to large-scale computing devices which were destined for long apprenticeships in computing installations where maintenance, environment, and general care were optimized. Secondly, the digital art had the advantage, during its early stages, of a much improved and more sophisticated electronic art. Digital devices now are beginning to appear in weapon systems, in process control applications, in air-defense installations, and in other places where extremely high performance and reliable operation are demanded. Although there is a background of knowledge bearing on reliability problems, and much remains to be learned, the digital discipline is in a much better position than was the analog discipline when it first entered into high-performance applications.

With respect to reliability, one might catalog digital systems in two ways: those for which the environment is chosen to suit the digital system, and those for which the environment is dictated by operational demands of some larger or other type of system. The first kind of environment may be called "optimum," with the typical computing installation for an example; and the second kind may be called "operational," with military or demanding industrial applications as examples. In the former case, a great deal is known about ways of providing long periods of trouble-free operation. Recent machine statistics indicate that the electronic parts of 5000-tube systems are exhibiting average times-between-errors of many hundreds of hours. In military and industrial applications, however, not much is known as to reliability. The environment is more rugged and the demands are higher; but in many cases the interval over which proper operation is required is much shorter. It is certain that digital devices will have some of the troubles that any electronic discipline must encounter when entering a new area, but this transition should be greatly eased as a result of the excellent achievements already made in optimum environment designs. It is for just this transition into military and industrial application that points made in the first paper have great significance.[1]

Boldyreff suggests that a component must be evaluated for reliability in its final environment, where this final environment must be understood to include the remainder of the system in its operational situation. Since the optimum-environment machines for the most part, are not parts of a larger system, but are themselves the entire system, this point has not to date particularly plagued digital designers. However, the airborne or seaborne digital computer either does or will have to face the problem. Its designers must not only think of all those things which designers of 704's, 1103's, JOHNNIAC's and other such machines thought of, but they must face a whole host of problems, some of which they may not appreciate or even know about until the first computer is put into its system environment. A computer, well-behaved in all laboratory tests for instance, might go completely beserk when operated in proximity to a source of electrical noise. Or some of the components of the computer may fail to survive unusual types of vibration.

A second point made by Boldyreff is that there must exist a compatibility between reliability and performance. Some levels of performance may demand extreme reliability from some part of a system, whereas a reallocation of performance requirements might greatly change the situation. For instance, an airborne computer as part of a weapon system may have the job of providing an appropriate course for intercept of a target. It is conceivable that, for the precision of control required, this machine would not have sufficient time to use some iterative loop a few extra times to offset a moment of weakness during which some part of the machine committed an error. However, with some increase in the lethal radius of the weapon proper, the closeness of control could conceivably be relaxed. Then the system could tolerate a more ill-behaved computer, since it would then have an opportunity to correct some of its difficulties through additional use of its error-reducing iterations.

There is here a germ of an idea which may have value. An analog computer when constructed and wired solves exactly one problem; parameters may be varied, but the problem may not. A digital computer, when wired, can solve, by adjustment of its routine, many problems or slight variants of the stated problem. Thus, some of the give and take in adjusting system performance can be accommodated by manipulation of the routine, even late in the development period. Further, the influence of the routine itself on the over-all reliability of the digital computer-containing system must not be overlooked. Many examples exist of routines which are normally well behaved, but which on certain special combinations of data input go completely wild. Thus, the programmer, as well as the digital design engineer, is likely to be an essential part of the team which is dedicated to providing digital computing devices for demanding applications.

Admitting then that the routine itself is part of the reliability problem of a digital system, it is also true that one way of experimenting upon a digital machine is through its routine. In the previous paper,[1] Hoffman showed how experiments can be designed to extract data from systems which are reluctant to yield concise statements of their performance or characteristics. One wonders to what extent a designer of experiments might prove valuable in constructing diagnostic routines, even though there is probably no college curriculum in the country that could produce this hybrid individual.

From this discussion, what might reliability mean to each of the several designers of any large system which contains a computer? To the component man, it means basic components such as capacitors or terminal boards or servo motors which meet certain tolerance specifications, which exhibit prescribed short- and long-term drifts, which can operate within specifications in a prescribed environment, and which have a specified and adequately small probability of catastrophic failure. The analog-components man is in a fairly advanced state of evolution, but the digital-components man has much to learn about the behavior of his components in adverse environments.

To the circuits man, reliability means circuits which use the parts supplied by the components man, which can accept input signals with a prescribed tolerance, which provide output signals to a prescribed tolerance, which tolerate specified supply fluctuations, and which also tolerate those aspects of the environmental situation which are not applicable at the components level. The digital-circuits man knows a great deal about designing for optimum environment, but he has much to learn about the operational environment. On the other hand, the analog-circuits man has learned the hard way and has evolved toward operational designs. He would, however, do well to backtrack now and to learn to make use of the conservative design philosophy which his digital brother has been forced to evolve. But his digital brother would also do well to hear what he has to say about the problems of meeting adverse environments.

Next in the reliability chain is the computer designer responsible for an over-all machine which he hopes will get the right answer each time, and which tolerates all of its environmental conditions—some of which may have appeared for the first time. However, the digital machine may not get the right answer each time, not because of electronic difficulties, but because the routine may malfunction on particular kinds of input data, or because the input data may be incorrect.

Lastly, therefore, is the man with the systems responsibility who must consider such interactions as this and guarantee over-all performance under the full environmental situation. Again some aspects of the reliability problem as influenced by environment may appear for the first time, such as mutual electrical interference problems. At this level must be considered such problems as, in the digital case, seeing that the gross job of the computer is done properly. This now means that the computer plus routine must be able to tolerate its environment, where the environment now includes such things as faulty input data and other misrepresentations of information.

Reliability means a completely different thing in detail at each of these levels and to each of these people, although the reliability at each level builds on the reliability of all which came before. Reliability in the complex system is not a simple thing. Considerable progress has been made with systems including analog computers. Much is to be learned concerning systems with digital computers, but the experience with optimum-environment digital machines has provided a good basis for advancement.

To sum up, a number of points have been suggested for consideration. The analog system, by virtue of its inherent logical feedback loops, can in many instances tolerate much that the digital system cannot; and in many applications the analog system need only exhibit reliability on the average. The digital system, because of its absence of redundancy, demands instantaneous reliability. This demand imposes new problems for the designer, since now short- and long-term drifts, initial tolerance, and catastrophic failure must all be considered. Reliability of any over-all design starts with a complete knowledge of component behavior, where, as Boldyreff has remarked, this behavior must be discovered in the final environment. Hoffman's techniques for optimizing the yield of an experiment should also be useful in learning the true statistical behavior of a component. From this knowledge, the circuit designer may then develop circuits which will operate under wide conditions. With these, the final machine and eventually the system can demonstrate its required reliability.

The digital industry to date has placed the bulk of its products in optimum-environment installations where careful maintenance, careful climatic control, and expert operation is routine. For these reasons, the digital art has had an opportunity to build highly reliable machines, which, for the most part, themselves constitute the entire system. However, new applications are appearing—such as military and certain industrial demands—in which the environment is neither optimum nor can the machine expect the care and maintenance of its computing-center predecessors. The analog machine has already made the transition from the sheltered laboratory to the world of real life, but even so it should profit from the methods of experimental design, the notions of large system reliability, and the new philosophy and techniques of the digital designer.

The digital field is fortunate in having seen the difficulties which the analog field has experienced; it is fortunate in having a more sophisticated and elegant electronic art to use. All of these things will help make its transition from the laboratory all the easier, but none of us should for a moment feel that all of the answers to reliability are known. It has been the purpose of this paper to suggest new viewpoints to old problems and to bring together some of the ideas about reliability which have never been documented. Although many of the principles of designing for reliability are clear, most of the details are not.

A Digital System Simulator

WILLIAM E. SMITH†

Introduction

Any computer can be described by logical Boolean equations and memory elements. The computer can then be constructed from physical components which realize these logical expressions and memory elements. Some of the more conventional components which are in use are semiconductor diode gates interconnected to represent logical equations, and various forms of the Eccles-Jordan flip-flop which provide binary storage.

The digital system simulator provides binary storage analogous to flip-flop storage in the form of magnetic cells on the surface of a rotating magnetic drum. The logical equations are also written onto the surface of the magnetic drum in a coded form. Then, a minimal number of diode gates and actual flip-flops are required to interpret the encoded logical expressions and pseudo-flip-flops, thereby causing the simulator to behave as the encoded digital system would behave. The size of the memory alone determines the complexity of the digital system which can be simulated.

† Aeronutronic Systems, Inc., Glendale, Calif.

Memory

In the prototype simulator, four nonvolatile channels, each extending completely around the magnetic drum, serve as the coded logic channels. These are the *J*, *K*, *N*, and *O* channels. See Fig. 1.

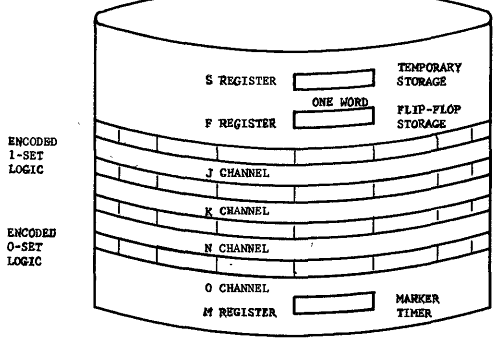


Fig. 1—The magnetic drum memory.

One magnetic drum recirculating register, the *F* register, which is $n+1$ bits in length, is used to store the states of n pseudo-flip-flops; one bit position is not used.

Another circulating register of the same length, the S register, is used for temporary storage of the new states of the pseudo-flip-flops as they are derived from the encoded logic. This register is also capable of precession in an n bit loop, which causes the contents to shift right one bit per word when necessary. A third circulating register of $n+1$ bits, the M register, is used to derive a marker or timing pulse at the end of each word.

Equipment Associated with the Memory

Each of the four nonvolatile logic channels, J, K, N, and O has a read amplifier and read flip-flop whose designations are J_r and J_1, K_r and K_1, N_r and N_1, and O_r and O_1, respectively. The read heads are also the write heads, which can be driven from the write amplifiers J_w, K_w, N_w, and O_w when initially filling the channels. See Fig. 2.

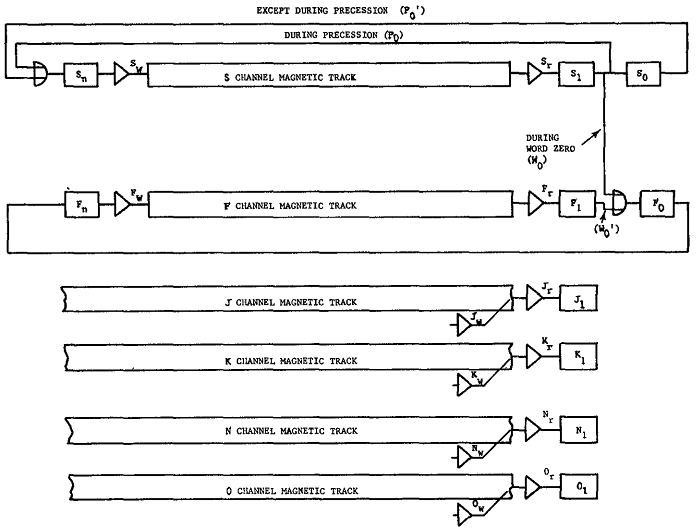


Fig. 2—Equipment associated with memory registers and channels.

The F register stores $n-2$ bits on the magnetic drum and three more bits in positions n, 1, and 0 by means of flip-flops F_n, F_1, and F_0. A read and write amplifier, F_r and F_w, complete the loop. The F register usually recirculates, but during word zero, W_0, when computing, it copies the contents of the S register.

$$
\begin{aligned}
F_0 \quad {}_1f_0 &= F_1W_0'C + S_1W_0C \\
{}_0f_0 &= F_1'W_0'C + S_1'W_0C \\
F_n \quad {}_1f_n &= F_0C \\
{}_0f_n &= F_0'C.
\end{aligned}
$$

The F register also serves a secondary function which is described in the filling procedure.

The S register stores $n-2$ bits on the magnetic drum, and the remaining three in flip-flops S_n, S_1, and S_0. It normally either precesses, P_0, in an n bit loop, or recirculates P_0' in an $n+1$ bit loop. New information may be written at the end of each word at time T_n.

$$
\begin{aligned}
S_0 \quad {}_1s_0 &= S_1C \\
{}_0s_0 &= S_1'C \\
S_n \quad {}_1s_n &= S_0P_0'T_n'C + S_1P_0T_n'C + T_n(\cdots). \\
{}_0s_n &= S_0'P_0'T_n'C + S_1'P_0T_n'C + T_n(\cdots).
\end{aligned}
$$

The M register similarly recirculates $n+1$ bits with the aid of three flip-flops, M_n, M_1, and M_0. A single one in position zero of the M register identifies the time T_n by its appearance in M_1; all other bits are zero in this register.

Word Structure and Coding

All words are described in normal form, as they would appeai in their appropriate registers at time T_0.

The F and S registers remember the present and next future states of the pseudo-flip-flops, respectively. Each bit position is assigned a flip-flop to represent, except bit position zero, which is not used. For convenience in this presentation, all pseudo-flip-flops are designated Q to avoid confusion with actual flip-flops. A word in the F or S register is then as shown.

Q_n	Q_{n-1}	Q_{n-2}	$\cdots$	Q_2	Q_1	/

This representation is always correct for the F register. It is initially correct for the S register (at T_0W_0) but the pattern will be shifted right during subsequent words, due to precession. Thus, after two precessions, Q_2 will be in position n at time T_0 in the S register.

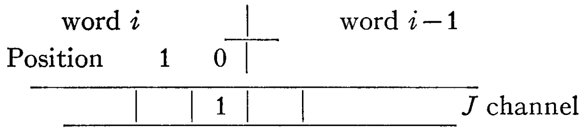
Q_2	Q_1	Q_n	$\cdots$	Q_4	Q_3	/

There are 64 words of $n+1$ bits each on the J, K, N, and O channels. A word may serve three functions.

Four ones in the zero position of word zero, W_0, in J, K, N, and O identify the word as word zero or the origin: accordingly, these set the word counter to zero.

word 00					word 77 (octal)	
Position	2	1	0	n	$n-1$	
			1			J
			1			K
			1			N
			1			O

A one in the zero position of any word on the J channel causes precession in the S register during that word time.



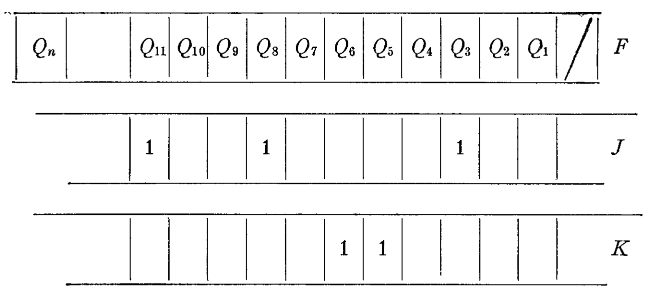
The P_0 flip-flop senses this code and controls precession.

$$P_0 \quad {}_1p_0 = J_1T_nC$$
$$\qquad {}_0p_0 = J_1'T_nC.$$

Note that the origin encoding will always cause precession during word zero.

The third and principal use of these channels is to encode and-gates. The encoding of an and-gate is accomplished as follows:

In a given word on the J channel "ones" are placed so that they will occur in coincidence with the propositions in the F register, which must be true to make the particular and-term true. On the K code channel "ones" are similarly placed in coincidence with the propositions which should be false, O, to make the same and-term true. Thus, the and-term $Q_{11}Q_8Q_6'Q_5'Q_3$ is encoded as shown:


Q_n
Q_{11} Q_{10} Q_9 Q_8 Q_7 Q_6 Q_5 Q_4 Q_3 Q_2 Q_1
F
1 1 1
J
1 1
K

Other and-terms are encoded in successive words.

An and-term always triggers the flip-flop simulated in the last bit of the word in the S register, which is Q_n in the previous example. Thus the complete logical equation is

$${}_1q_n = Q_{11}Q_8Q_6'Q_5'Q_3.$$

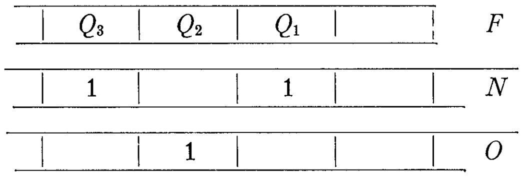
This also implies that if no precession in the S register is called for between successive words, successive and-terms will trigger the same flip-flop, so that these and-terms are effectively or-ed together. If precession is called for, the and-term will affect the next flip-flop in succession.

By the proper use of the precession code all flip-flops can be caused to obey their logical input equations in sequence.

The J and K channels are used to encode 1-set terms. The N and O channels are similarly used to encode 0-set terms. For example, the term

$${}_0q_n = Q_3Q_2'Q_1$$

would be encoded:



And-Gate Simulation

To obtain the function of an and-gate the coding must be properly interpreted. A flip-flop D_1 determines whether or not the coded and-gate is true in the following manner:

The flip-flop D_1 is initially set true at the start of each word.

$$D_1 \quad {}_1d_1 = D_1'T_nC.$$

Then, as a word is read serially, if a code 1 in channel J is not coincident with a true proposition (a 1 in channel F), or if a code 1 in channel K is not coincident with a false proposition (a 0 in channel F), the flip-flop D_1 is reset. All conditions for the and-gate to be true have not been met.

$$D_1 \quad {}_0d_1 = (J_1F_1'T_n' + K_1F_1T_n')W_0'C.$$

Flip-flop D_1 is sensed at the end of a word. If it is still true, the and-gate is true. The J and K channels accomplish the 1-set triggering of the last simulated flip-flop in the S register. Thus, at the end of a word, if D_1 is true, the last bit in the S register is set to a 1.

$$S_n \quad {}_1s_n = S_n'D_1D_0'T_nC.$$

Two similar channels, N and O, by an analogous process, may instigate the 0-set triggering of the same flip-flop. D_0 assumes the role of D_1 in this analogy.

$$D_0 \quad {}_1d_0 = D_0'T_nC$$
$$\qquad {}_0d_0 = (N_1F_1'T_n' + 0_1F_1T_n')W_0'C$$
$$S_n \quad {}_0s_n = S_nD_0D_1'T_nC.$$

This particular prototype does not allow for simultaneous 1-set and 0-set logic as might occur when using *jk* flip-flops. The type of flip-flop which it simulates is the set-reset or *rs* flip-flop, which can receive either a 1-set or 0-set trigger, but not both, during the same iteration of the logical equations.

The next word in each code channel represents another and-gate and may similarly instigate the 1-set or 0-set triggering of the same simulated flip-flop. Thus, these two and-gates are functionally or-ed together since they affect the same flip-flop. The extension of this concept to any number of and-or gates which affect the same flip-flop is obvious.

If successive one-word and-gates do not apply to the same simulated flip-flop, a 1 code in the zero position of the J channel causes the and-gate encoded in this word to trigger the next simulated flip-flop in sequence by causing the S register to precess. To be consistent, the first and-gate computed, encoded in W_0, contains a precess code (part of the origin code) so that flip-flop Q_1 is the first flip-flop triggered, followed by Q_2, Q_3, etc., rather than Q_n followed by Q_1, Q_2, etc. This precess code transfers triggering to the next flip-flop by causing the significant contents of the S register to precess one bit position to the right each word-time that the code is employed.

Transfer of New Data

After one revolution, *i.e.*, one solution of the complete set of logical equations, all and-gates have been sampled and all flip-flops have been set to their new values in the S register. The J channel must have been coded for $n-1$ precessions so that the S register is now in normal form.

It remains to transfer this new information from the S register to the F register in preparation for the next iteration of the equations. This occurs during word zero, W_0.

$$F_0 \quad {}_1f_0 = S_1W_0C$$
$$ {}_0f_0 = S_1'W_0C.$$

In order that gating with new values can proceed during word zero, the D_0 and D_1 flip-flops must sample the new values (not yet available in F_1) in S_1 during W_0.

$$D_1 \quad {}_0d_1 = (J_1S_1'T_n' + K_1S_1T_n')W_0C$$
$$D_0 \quad {}_0d_0 = (N_1S_1'T_n' + O_1S_1T_n')W_0C.$$

Note that this implies that the initial values at the start of computation be set up in the S register, and that the F register initial values are immaterial.

Origin Fill

The origin key α causes a one to be set into each of the four code channels, J, K, N, and O at the same time by the following means: a flip-flop B_1 is set on by the α key for just one clock period

$$B_1 \quad {}_1b_1 = \alpha B_1'I_1'C$$
$$ {}_0b_1 = \alpha B_1C.$$

An interlock flip-flop I_1 prevents B_1 from being set on more than once while the origin fill key is held down. See Fig. 3.

$$I_1 \quad {}_1i_1 = \alpha B_1C$$
$$ {}_0i_1 = \alpha'C.$$

While αB_1 is true, a one is written into each of the four code channels. At other times nothing is written

$$ {}_iJ_w = \alpha B_1$$
$$ {}_iK_w = \alpha B_1$$
$$ {}_iN_w = \alpha B_1$$
$$ {}_iO_w = \alpha B_1.$$

The four code channels must be cleared first by pressing the code channel clear key, γ, to erase any previous origin indication.

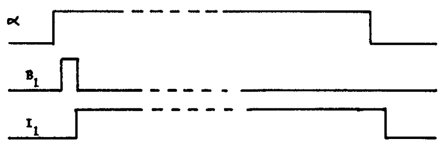


Fig. 3—Origin fill waveforms.

$$ {}_iJ_w' = \gamma$$
$$ {}_iK_w' = \gamma$$
$$ {}_iN_w' = \gamma$$
$$ {}_iO_w' = \gamma.$$

The origin is then indicated by four simultaneous ones in J_1, K_1, N_1, and O_1.

$$X_0 = J_1K_1N_1O_1.$$

M and F Register Fill

The origin pulse, X_0, is copied into the M register in position zero when the ϕ key is depressed. At other times the M register recirculates.

$$M_0 \quad {}_1m_0 = X_0\phi B_1C + (\phi B_1)'M_1C$$
$$ {}_0m_0 = M_0C.$$

Flip-flop B_1 causes switch ϕ to be effective at the proper time.

$$B_1 \quad {}_1b_1 = \phi X_0B_1'I_1'C$$
$$ {}_0b_1 = \phi X_0B_1C.$$

Flip-flop I_1 acts as an interlock to prevent further writing in the M register until after the ϕ key has been released.

$$I_1 \quad {}_1i_1 = \phi B_1C$$
$$ {}_0i_1 = \phi'C.$$

The origin pulse, X_0, is also copied into the F register but in position one when the ϕ key is depressed.

$$F_1 \quad {}_1f_1 = X_0\phi B_1C + (\phi B_1)'F_rC$$
$$ {}_0f_1 = F_1C.$$

At other times the one digit in the F register recirculates.

The F and M register fill waveforms are shown in Fig. 4.

J, K, N, and O Channel Fill

When the control panel selection switch is set to JK and one of the fill keys 0, 1, 2, or 3 is depressed, the J and K channels are filled simultaneously in word W_i and in the position specified when $F_1=1$.

$$J_w \quad {}_iJ_w = (2+3)JKF_1B_1W_i$$
$$ {}_iJ_w' = (0+1)JKF_1B_1W_i$$
$$K_w \quad {}_iK_w = (1+3)JKF_1B_1W_i$$
$$ {}_iK_w' = (0+2)JKF_1B_1W_i.$$

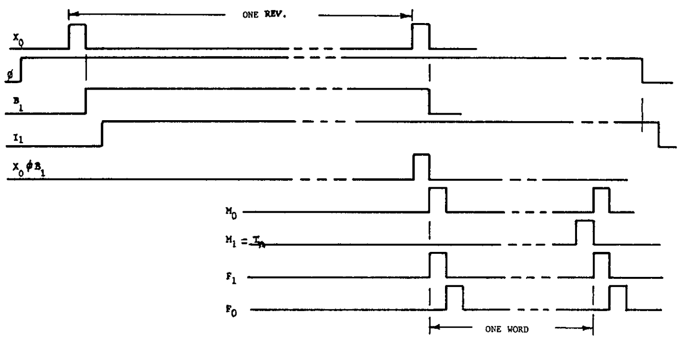


Fig. 4—*M* and *F* register fill waveforms.

The fill keys 0, 1, 2, and 3 are decoded as indicated in the logical equations above and also as indicated in the following table.

Fill key	J	K
0	0	0
1	0	1
2	1	0
3	1	1

The proposition W_i is logically derived from switches which sample the word counter for count i.

$$W_i \text{ (switch logic)}$$

Flip-flop B_1 causes the fill keys to be effective at the proper time.

$$B_1 \quad {}_1b_1 = \lambda T_n B_1' I_1' C$$

$$ {}_0b_1 = \lambda F_1 W_i B_1 C$$

where

$$\lambda = 0 + 1 + 2 + 3.$$

Flip-flop I_1 prevents another digit from being filled until after the currently used fill key is released.

$$I_1 \quad {}_1i_1 = \lambda B_1 C$$

$$ {}_0i_1 = \lambda' C.$$

As the selected digit is written into memory at time $(\lambda F_1 W_i B_1) = 1$ the 1 code in F_1 is delayed there by not allowing it to be erased to zero,

$$F_1 \quad {}_1f_1 = \lambda F_r C$$

$$ {}_0f_1 = (\lambda F_1 W_i B_1)' F_1 C,$$

and not allowing it to be copied into F_0,

$$F_0 \quad {}_1f_0 = (\lambda F_1 W_i B_1)' F_1 C$$

$$ {}_0f_0 = F_1' C.$$

Therefore, when the next digit is to be filled, $\lambda F_1 W_i B_1$ will be true one digit time later and cause filling of the next successive digit. This term is true at time T_0 just after the marker has been written, when digit position one is to be filled. It is finally true at time T_n when digit $n+1$ (which is digit zero of word $i+1$) is filled.

At the time that this last position is filled, the 1 in F_1 is immediately erased and not copied into F_0; therefore the F register is cleared to zero and no further digits will be written until the marker is again written in position one.

$$F_1 \quad {}_0f_1 = (\lambda F_1 W_i B_1) T_n F_1 C.$$

Also, during word 77 when the single one reaches position n in F_1 the F register is cleared to zero by not copying it into F_0, thus preventing filling in the origin position.

$$F_0 \quad {}_1f_0 = (\lambda F_1 W_i)' F_1 C + B_1' (W_{77} T_n)' F_1 C.$$

The N and O channels are similarly filled when the fill selector switch is set to *NO*,

$$N_w \quad {}_iN_w = (2 + 3) NOF_1 B_1 W_i$$

$$ {}_iN_w' = (0 + 1) NOF_1 B_1 W_i$$

$$O_w \quad {}_iO_w = (1 + 3) NOF_1 B_1 W_i$$

$$ {}_iO_w' = (0 + 2) NOF_1 B_1 W_i.$$

Operation

In operation, the simulator is first filled with the appropriate data to solve the particular problem at hand. A logical designer must reduce this problem to logical equations and thence to *J*, *K*, *N*, and *O* channel coding. To fill these channels the "Operation Mode" switch is set to *JK* or *NO* and the "Word Selector" switches are set to the word to be filled. See Fig. 5. The marker button "ϕ" is depressed to establish the beginning of a word, and then buttons "0," "1," "2," or "3," are depressed as required by the coding. Each entry causes the next successive character to be recorded.

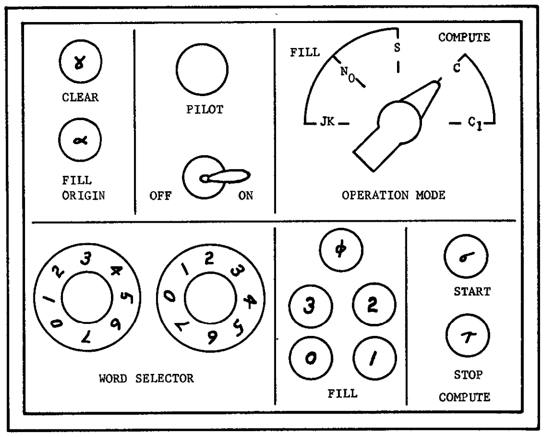


Fig. 5—Sketch of control panel.

After all logic is encoded, the "Operation Mode" switch is set to *S* and the initial setting of the flip-flops to zero or one in the *S* register is undertaken by depressing either the "0" or "1" buttons, respectively; with the word selector switch set to the appropriate word, namely, to the word with the jth precess pulse in it, the pseudo-flip-flop Q_j is filled.

The simulator is now ready to compute. To start computing set the "Operation Mode" switch to C and depress the "start" button. To stop the computer depress the "stop" button or wait for a programmed stop caused by the logical term $D_1D_0T_n$. If the "Operation Mode" switch is set to C_1 the computation will automatically stop after each drum rotation.

Output is obtained by oscilloscope or by logical gating plugs which select the desired output from a particular flip-flop.

Use

It is intended that the first prototype simulator be used as a logical designer's aid in designing and checking out equipment; also, as a training device for logical designers. Of course, many simple "games" and slow control problems may be coded into this simulator.

The speed of solution of a problem is relatively slow; one iteration of the complete set of logical equations takes one drum revolution. The number of flip-flops which can presently be simulated is about 30 and no simulated internal memory is available. These disadvantages will be relatively easy to obviate by logical means in the second prototype except for the relatively slow computation rate which is, however, adequate for many slow real-time control problems such as traffic control, production control, or logistics.

Acknowledgment

I would like to acknowledge the interest and cooperative effort shown by Dr. T. A. Rogers of the University of California, Los Angeles, and C. L. Wanlass and W. I. Pyle of Aeronutronic Systems, Inc., who have been instrumental in providing for the current construction of two simulators, one at UCLA and one at Aeronutronic Systems, Inc.

Discussion

R. R. Johnson (General Electric): Are you going to offer the services of this computer for checking out equations, and how large a computer can it simulate, and how can you check the simulated results?

Mr. Smith: Possibly. This particular model can simulate a computer of 28 flip-flops in something like 128 gates. However, we are thinking seriously of extending by simply putting more J, K, N, and O channels on this so that the number of gates would be doubled. There will be this memory feature in the computer and we can adjust that for various word lengths in your memory.

W. R. Smith (Datamatic): Have you tried to use this computer to reduce the amount of equipment in the simulated computer?

Mr. Smith: I have thought of that, but it seems you need the same amount of equipment to interpret the logic that you put into the computer.

A. Dinkel (Convair): What is the major difference between this and, say, Cash Register Corporation's DDA?

Mr. Smith: This is a machine for processing logic. The DDA is a machine which essentially adds an overflow. They perform two different functions. In addition, an overflow can be shown to be an integration. The DDA does additional integrations and the simulator processes only logic for you, whereas, the GP has a computing center which processes methodic operations.

W. J. Willis (Atomics International): What sort of selling price can be foreseen for school labs?

Mr. Smith: This first one is costing us around $12,000.

A New Input-Output Selection System for the Florida Automatic Computer (FLAC)

C. F. SUMMER†

Introduction

THE ULTIMATE goal of the original FLAC designers was to produce a fully coordinated instrumentation system and data reduction system with a common data language. Their foresight has resulted in a coordinated data gathering and data processing system, capable of many evolutionary changes as the "state of the art" progresses.

Radar data is presently collected in the field, on punched paper tape, in a language which the computing facility (FLAC) is capable of handling directly without intermediate processing. Success has been attained in recording field data from a classified electronic system directly on magnetic tape. This recording, in digital form, can immediately be processed by the FLAC without any preliminary handling.

Data also gathered by either photographic or electronic means from several geographic sources on the test range are all correlated by the central timing system. If 2 cinetheodolites are employed in a net, then 2 separate punched paper tapes are derived from the photographic data. Using 2 high-speed punched paper tape readers, the data collected from these sources can be collated, with time, by the FLAC at computer speeds.

Basic position data derived from the mathematical solution for a 2-station theodolite net is usually readout to 1 paper tape punch, velocity components are computed and readout to a second punch, acceleration data is computed and readout to a third punch, and other data items such as first and second differences or mathematically smoothed quantities can be readout to other devices. This entire processing cycle, from multiple readins to multiple readouts, is accomplished with one "data pass" through the computer. A tremendous time saving in data processing can thus be realized.

Present and Future Input-Output Capability

It becomes obvious from the example cited, that the computing system, for the reduction of large quantities of missile test data, in a timely and efficient manner must be capable of communicating with a wide variety of input-output devices; all under control of the computer program. This philosophy was inherent to the original logical design and therefore the input-output addresses of FLAC can be made very extensive.

† RCA/Missile Test Project, Patrick Air Force Base, Fla.

Over the past three years the quantity and quality of input-output devices has increased; from a single Flexowriter and magnetic wire readin units to a full complement of devices. In the expansion of the number and types of input-output devices, it has been necessary to add the new high-speed input-output selection system to complete the computer input-output communication and control link.

Future data acquisition plans point to a requirement for at least the following computer input-output devices:

1) Four magnetic tape input-output handlers—4 addresses.
2) Three high-speed punched paper tape readers—3 addresses.
3) Three paper tape punches (60 characters per second)—3 addresses.
4) Two paper tape punches (120 characters per second)—2 addresses.
5) One medium-speed punched paper tape reader (60 characters per second)—1 address.
6) One Flexowriter—1 address.
7) A magnetic-tape bin memory (400,000 words with a 60-sec mean access time to any word)—20 addresses. (10 additional addresses for rewind orders.)

Theory of Operation of System

To appreciate the capability and versatility of the FLAC with regard to multiple input-output devices under control of the computer, a brief description of some of the pertinent machine characteristics seems desirable.

The modified FLAC is a series-parallel machine patterned originally after the SEAC and the MIDAC systems. The remaining similarity is only in the type of logical circuitry employed. The arithmetic unit is composed entirely of dynamic logic type circuitry; 1 tube type (computer grade 6AN5), one type of pulse transformer, and 10 resistor values. A new 4096-word magnetic core memory handles the binary information in a parallel fashion and requires special circuitry to mate it to the arithmetic unit (au) and center controls. A FLAC word is composed of 44 binary digits, plus sign. The pulses are timed at a 1-megacycle rate and the basic computer timing is controlled by 4-phase 1-megacycle source (computer clock).

The operational timing unit of FLAC is the "minor cycle" which is 48 μsec long. Thus, the computer word utilizes 45 one-μsec pulses and there are 3 "blanks" which are involved in certain logical operations.

The computer is a 3-address machine cycled through 4 operational phases (F1, F2, F3, F4), possessing 16 basic orders or commands. Some special features are: complete binary-to-decimal and decimal-to-binary number conversion (as an order, not a subroutine), a tally order which controls a base counter. Addresses can be made relative to the base counter or control counter by the special control group of the instruction. Readin and readout orders make use of a special word counter which allows from one to 4096 words to be read in or out with only 1 instruction. The readin or readout then makes addresses relative to the word counter as well as control or base counter.

Input-Output Selection System Logic

Construction of Computer Word

A computer word can be used to represent either instructions or data. The composition of the instruction word is such that 12 binary digits of the word are devoted to each of the address groups: alpha, beta, and gamma. Four digits are for the operation code and four digits are for the control group. The pulse positions of the instruction word are as follows:

P_{46}–P_{48}	P_1	P_2–P_5	P_6–P_9	P_{10}–P_{21}	P_{22}–P_{33}	P_{34}–P_{45}
Blank	Sign	Opn Code	Control Group	Gamma	Beta	Alpha

Transfer and storage of words are made in a serial form throughout the arithmetic unit and central controls.

Input-Output Address System

It is convenient within the computer and for programming purposes to express the numbers utilized by FLAC in the hexadecimal system. In this case 4 binary digits are represented by one hexadecimal character. Thus, an input-output address can be represented by "01C" (a high-speed paper tape reader). In binary form it is reduced to 0000 0001 1100. The following is an example of the hexadecimal addresses of the various FLAC input-output devices:

01B, 01C, 01D—The 3 high-speed paper tape readers.

Instruction Storage-Loop Timing and Logic

The instruction storage loop (isl) is a 48-μsec recirculation loop. This loop has several access points in order that instructions contained therein may be sampled by other units of central control (*i.e.*, operations staticizer, run-halt controls, address selector, and relative-address flip-flops).

An instruction is transferred from the memory or shift register to the isl during the first minor cycle of phase 1 (F1). This instruction then remains in the loop (recirculated) until the next phase 1 (F1), at which time it is erased (recirculation inhibited) as the new instruction is transferred in. Gates are available for erasing the contents of the loop when desired by means of a switch on the control console.

The position of a pulse being recirculated within the isl can be referenced in time or loop position for logical purposes. For example, the basic time (T) of a single pulse may be expressed as T_{22} which essentially means that it is the 22nd pulse time of the particular minor cycle under consideration. Loop time (L) is also considered in the timing logic as well as pulse position (P) in a particular minor cycle. It is convenient to relate these three time elements (T, L, and P) in the following expression:

$T-L=P$. Thus at $L=0$, L_{22} will be present at T_{22} of a particular minor cycle. If $L=3$, P_{19} will be present at T_{22}.

Fig. 1 (opposite) presents the basic logic of the input-output selector. There are shown gating circuits which are connected to the $L=22$ and $L=34\frac{1}{2}$ access points in the instruction storage loop. From the logic it is shown that the first pulse of beta (β) is present at $L=22$ at the input to that particular "AND" gate, but delayed 0.75 μsec.

The first pulse of Gamma (γ) is present at $L=34\frac{1}{2}$ at the input to the other "AND" gate, but also delayed 0.25 μsec. T_i, T_o, T_m and ($\overline{\text{Manual}}$) (termed "Manual Bar" indicating absence of manual condition) are conditions on the particular gates shown for proper actuation. The outputs of both "AND" gates are buffered in an "OR" gate which is clocked at phase 4 (CP4). Either gate output then is present for a readin or readout on the buss as shown.

Shown also in Fig. 1 are 8 "AND" gates timed at T_6, T_5, T_2, T_1, T_{48}, T_{47}, T_{46}, T_{45}. Each gate is also checked at CP_1 (Clock phase 1). The outputs feed 8 amplifiers and also pulse stretching one-shot multivibrators. The purpose of the multivibrator is to lengthen the computer pulses which appear at each input; the average output voltage actuates the plate-circuit relay schematically shown as (K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8). Relays, gates, etc., pertaining to K_1 and K_2 are associated with the computer electronic format circuitry and are not within the scope of this paper. As long as an instruction (readin or readout) is circulated in the isl, pulse patterns will appear on the output buss from the isl. K_3, K_4, K_5, K_6, K_7, K_8 are each activated by a particular binary pulse pattern, depending upon the binary number associated with the address of a particular input-output device. Thus, the 8 relays form a "tree" feeding an 8×8 matrix which is capable of making 64 different selections. These relays are termed "decoding" relays.

To illustrate the selection process, the input-output address 001 (as represented in hex) is used. 001 is the specific input-output address of the Flexowriter. The binary pulse pattern for 001 will be 0000 0000 0001 and is gated into the isl where it is recirculated. Fig. 2 indi-

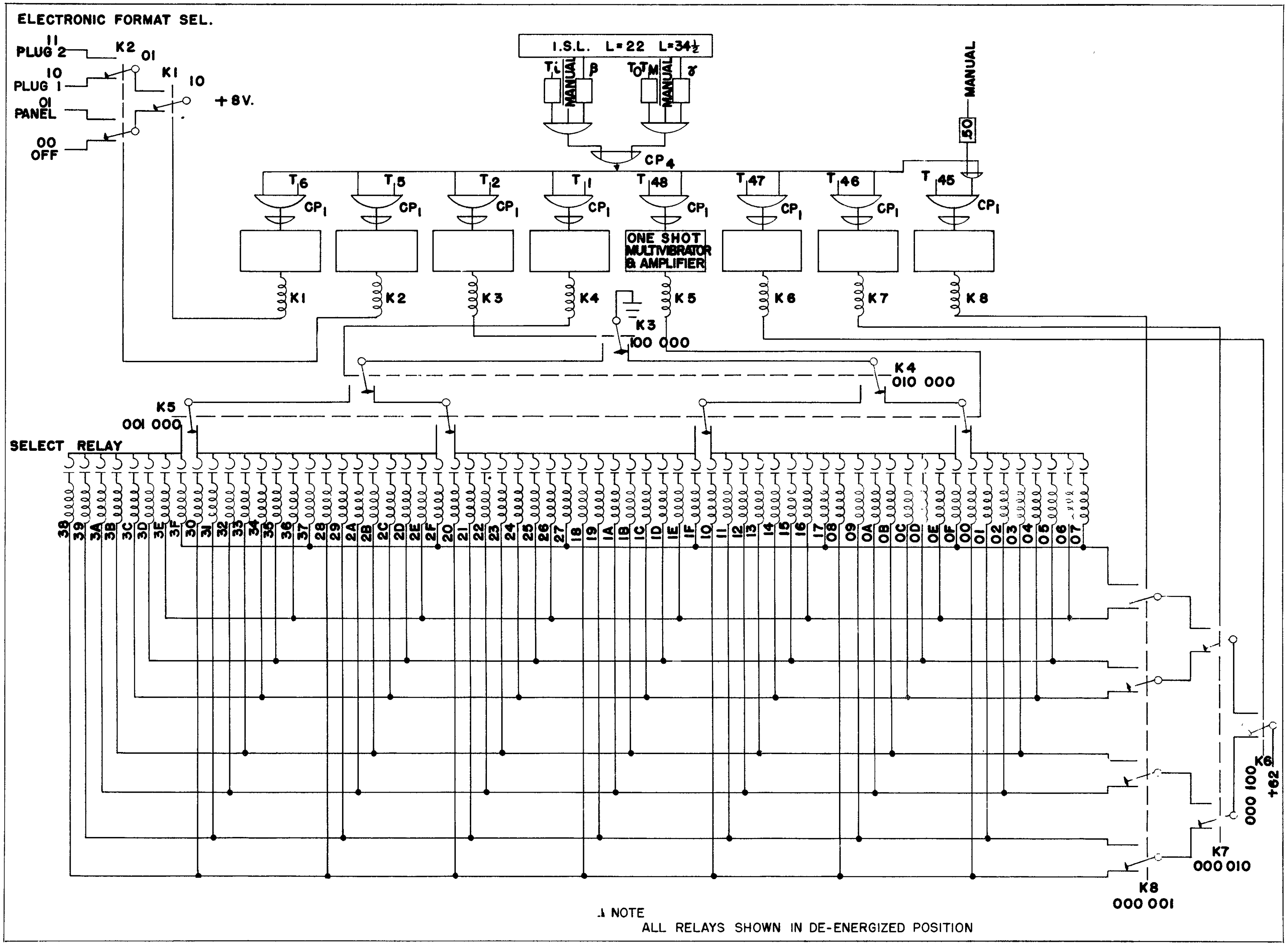


Fig. 1—FLAC input-output selection logic.

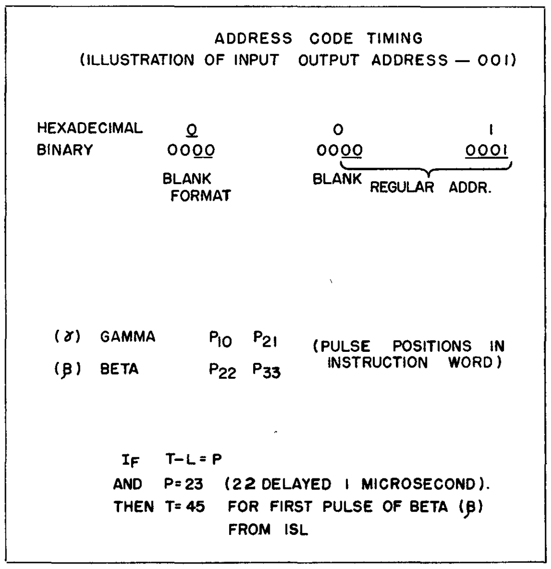

ADDRESS CODE TIMING
(ILLUSTRATION OF INPUT OUTPUT ADDRESS — 001)
HEXADECIMAL 0 0 1
BINARY 0000 0000 0001
BLANK FORMAT
BLANK REGULAR ADDR.
(γ) GAMMA P10 P21
(β) BETA P22 P33
(PULSE POSITIONS IN INSTRUCTION WORD)
IF T−L = P
AND P=23 (22 DELAYED 1 MICROSECOND).
THEN T=45 FOR FIRST PULSE OF BETA (β) FROM ISL

Fig. 2—Input-output address code timing.

cates the portion of the computer word which the address chosen actually occupies in time. As the first pulse of beta (β) (for a readin order) arrives at the K_8 position, it can be seen from the timing illustration in Fig. 2 that a T pulse (T_{45}) will also be present on the other input to that gate. Conduction will occur as soon as clock phase (CP_1) comes up and K_8 will be energized. By circulating the remaining digits of the selection code (in this example they are all "0's"), it should be apparent that all other digits will arrive at the particular gate associated with the proper "T" time. However, the relays do not energize since the gate-input conditions ("0's" in the remainder of the address) satisfy the proper time relationship.

The circuitry shown in Fig. 1 will indicate that the K_8 relay activated and the conduction path through the matrix selection relay designated "01" was completed. This matrix relay (or relays as is the actual case) picks up and in turn completes the following circuits to the input-output device 001 (Flexowriter) (these are not shown in Fig. 1):

1) Information lines "1" "2" "3" "4" "5" from the input-output shift register.
2) $e-1$, a signal remaining on until the proper number of words have been transferred.
3) Flexo-sync, a signal which permits $e-1$ to exist on readouts and synchronizes the Flexowriter to the computer.

Check Circuitry

To give a positive indication of malfunction such as a selection of more than one address (unit) etc., a circuit was designed to detect an indication of this fact. It has been aptly designated as the "only one" circuit, since it indicates the correct selection of "only one" unit. Fig. 3 (opposite) presents a typical "only one" circuit. The contacts shown connected to −65 v are contacts of the select relays in the matrix. As long as only one contact is closed there is a single 5.1 k resistor clamped (via the diodes) to −10 v. If more than one contact closes then the current drawn will be proportionately greater, thus essentially biasing the following stage below proper operating level. There is a 4.2-k "pull up" resistor connected to +62 v and also clamped to +2 v. This line is connected to the check circuit "AND" gate as shown. The phase 2 (F2) signal drives a one shot multivibrator which is cathode coupled to the "only one" line. This is adjusted to provide approximately 20 to 30 milliseconds of delay. The delayed "only one" signal also appears elsewhere in the computer as positive "AND" gate condition for selection of readin or readout functions. It is necessary to provide this delay in order to allow switching transients to disappear before input-output selections are actually made in quick succession. Various conditions are shown connected to the "check circuit" gate which will recirculate every minor cycle until inhibited by $\overline{T}_{45}$ (indicating the absence of T_{45}). The output of the following stage indicates correct selection. This signal also provides an inhibit on the "wrong selection" light gate as shown. Thus, "correct selection" and "wrong selection" cannot be "on" simultaneously unless a malfunction has occurred.

Selection-Matrix Relays

A most interesting element of this new input-output selection system is the type of relay utilized in the selection matrix. Specifications for this relay are generally as follows:

1) Operating time—approximately 3 milliseconds.
2) Hermetically sealed contacts which are individually replaceable.
3) Average operating current, approximately 10 ma (20 milliwatts).
4) Reliable operating cycle—at least 4 million operations.

This relay utilizes the "Glaswitch" contact element developed by Western Electric Company, and was manufactured by Revere Corporation under Western Electric Company license.

Fig. 4 shows a scaled photograph of a single replaceable "Glaswitch" element, a single relay unit manufactured by Revere Corp., and a 3-unit relay package designed and fabricated locally for this particular application. If less than 3 units are in parallel, a series resistor must be incorporated for proper operation in the matrix. The diodes are also mounted in the base of the relay package and are used to prevent multiple-current-condition paths to other relays and thereby cause malfunctions. Fig. 5 presents a front view of the decoder panel with the one-shot multivibrators and the decoder

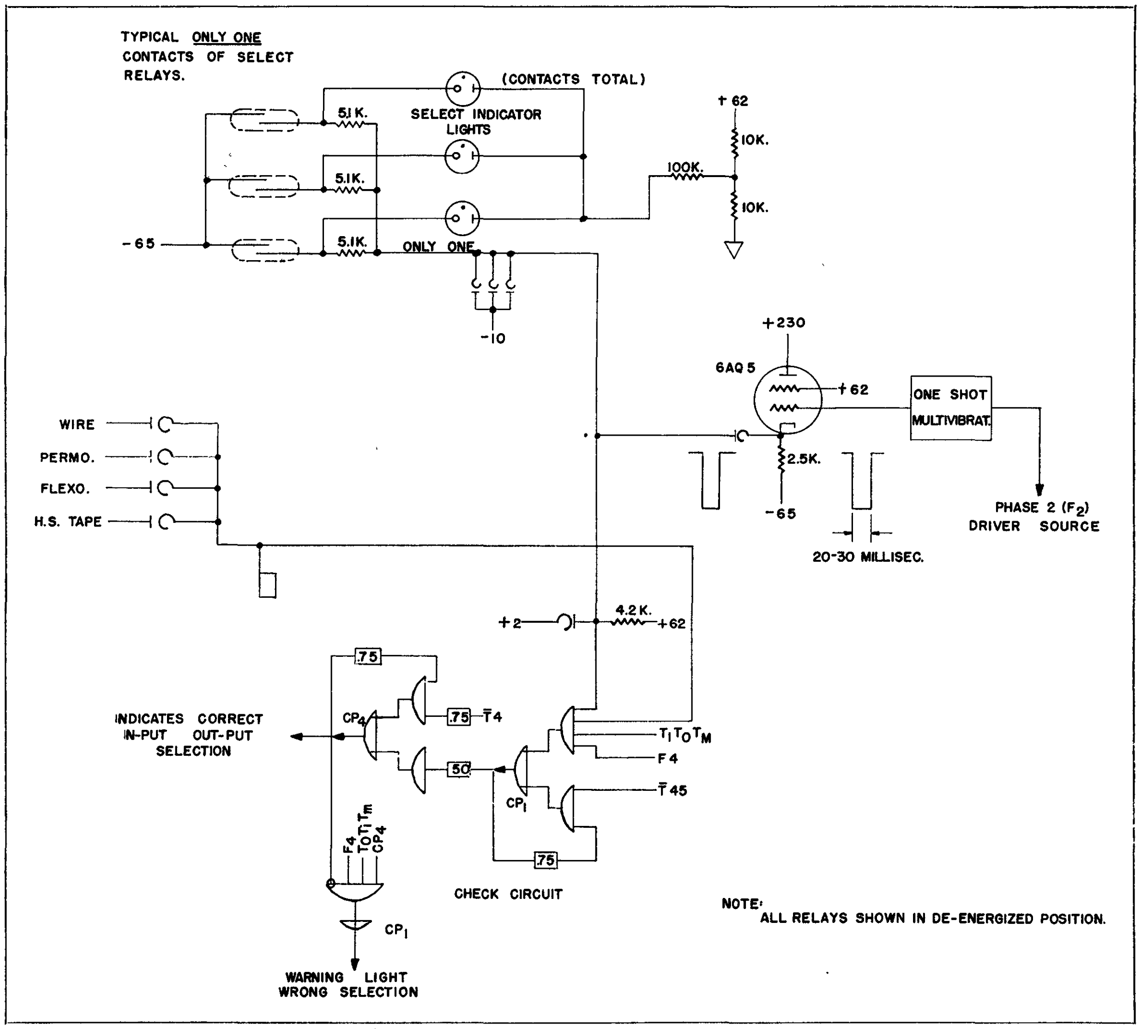


Fig. 3—Logical diagram—typical "only one" circuit.

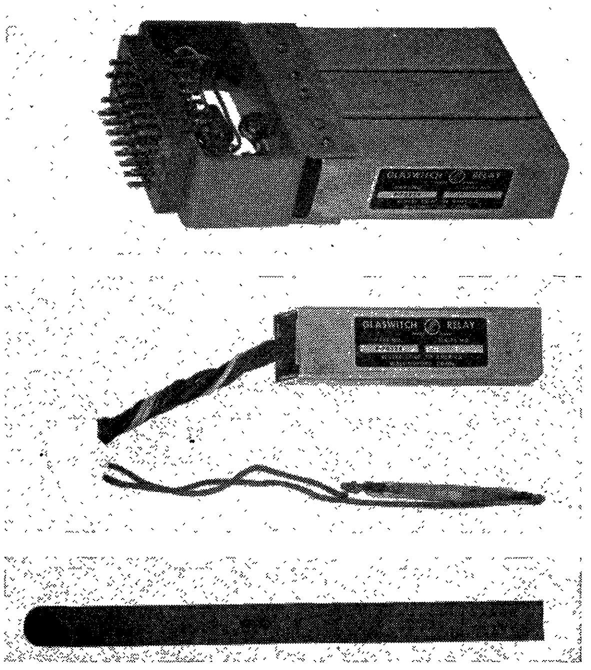


Fig. 4—Relay package and components, selection matrix.

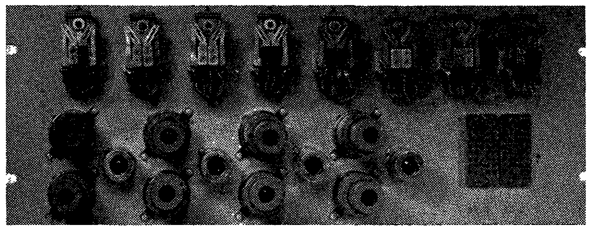


Fig. 5—Decoder chassis—front view.

relays, K_1 through K_8. This panel is mounted within the computer control console. Connectors for signal lines and power circuits are shown. Fig. 6 presents a select relay "stair step" for a total of 50 different addresses. The available space dictated somewhat the peculiar construction and layout. The bottom row of sockets connect to various input-output units as indicated. This unit is also mounted in the computer control console.

Conclusion

The multiple input-output system described has been in operation on the FLAC for approximately four

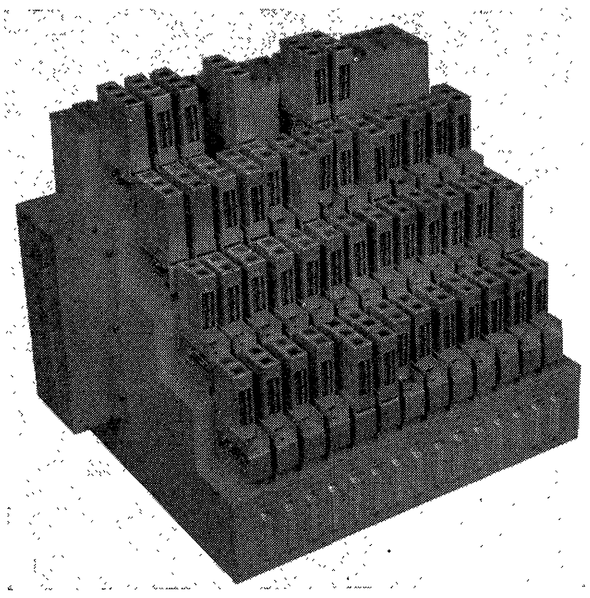


Fig. 6—Selection matrix—relay stair step.

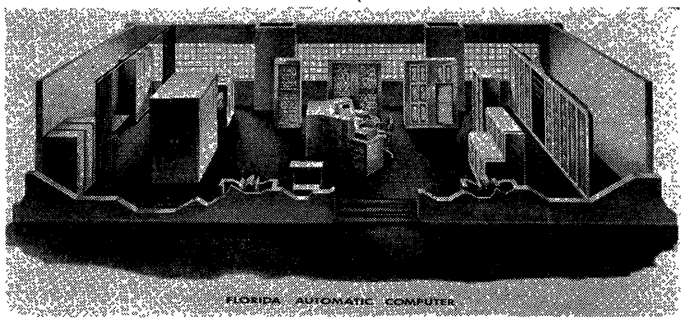


Fig. 7—Florida automatic computer—FLAC 1.

months. Service has been excellent with no relay failures being evident in the select relay matrix. The "Glas-witch" relay has proven to be a highly reliable device for high-speed switching of multiple input-output devices under programmed control of the computer.

This system permits a wide variety of input-output devices to be addressed and controlled by the FLAC and permits the reduction of massive quantities of missile test data from a number of sources in a reliable, efficient, and timely manner. Fig. 7 presents an artist's conception of the entire FLAC.

ACKNOWLEDGMENT

Recognition should be made of R. W. Mitchell and J. J. Schell whose engineering design work has made this system possible, and to those laboratory personnel who gave such excellent cooperation and assistance in the construction phases of this project.

Discussion

R. R. Johnson (General Electric): Just what are the input-output specifications on data flow rate, etc., and how does this contrast with the computer speed?

Mr. Summer: The units control the rate of flow of information in and out. The paper tape has an input speed of about three hundred characters per second, and the output is about sixty characters per second on the punches.

A. B. Crawford (Signal Corps, Ft. Hauchuca): Are any of your multiple input-output units remote, and is the FLAC able to compute simultaneously with input?

Mr. Summer: None of the units are remote.

R. A. Jensen (IBM): Are the punched paper tape punches located at the remot- data gathering equipments, or is the digie talized data transmitted from the theodolites to punches located at the FLAC computer?

Mr. Summer: Currently, the theodolites. The next step is to digitalize the position in place on magnetic tape. In the case of radar, it is digitalized in the punch paper tape or perforated at the radar site. Eventually, this also will go to magnetic tape.

The IBM 650 RAMAC System Disk Storage Operation

DAVID ROYSE†

Disk Storage Operation

THE IBM 650 RAMAC System combines the computing flexibility of the Type 650 magnetic drum data processing machine with the quick random-access large-scale memory which may be assembled from several of the Type 355 disk storage units. The combination may be programmed to perform rapid sophisticated jobs of in-line (single-step) data processing. Briefly, this means that the punched-card or taped record of each event is in turn completely processed against all of the records, inventories, or summaries which it affects. The various physical units which may comprise the system are shown in Figs. 1-3.

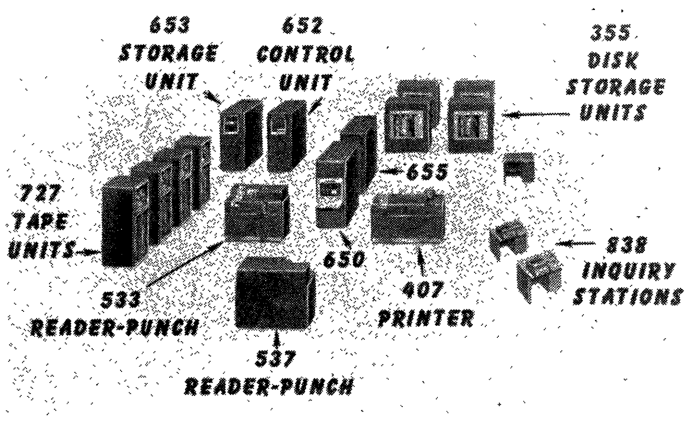


Fig. 1—IBM 650 RAMAC.

Basic Computer

The basic computer consists of the elements shown in Fig. 2: from one to three of the three card inputs; card or printer output units may be had in any combination, each with or without alphabet. The 655 unit contains the power supply for itself and for the 650 console unit. It also contains input-output translating, and any associated alphabetic equipment. The 650 console unit contains the magnetic drum main memory and the principal arithmetic, logical, and timing elements; it has the display console. (See Fig. 4, next page.) The 650 is the nerve center of any of the expanded 650 systems which may be assembled.

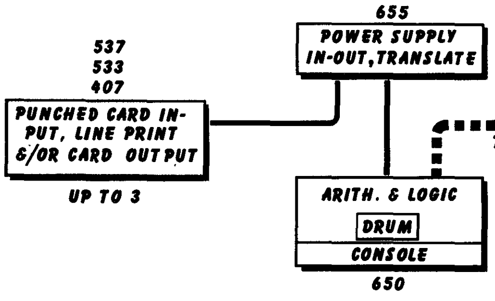

655
POWER SUPPLY IN-OUT, TRANSLATE
537
533
407
PUNCHED CARD INPUT, LINE PRINT &/OR CARD OUTPUT
UP TO 3
TO 653
ARITH. & LOGIC
DRUM
CONSOLE
650

Fig. 2—The basic 650.

The basic computer is a stored program, single address, intermediate speed machine. It is a serial by digit parallel by bit machine with 125-kilocycle clock rate. It has a main memory capacity of 2000 ten-digit words, a very comprehensive list of commands (including automatic table-look-up), and it is very easily programmed.[1] Circuit design is conservative and the machine is thoroughly self-checked.

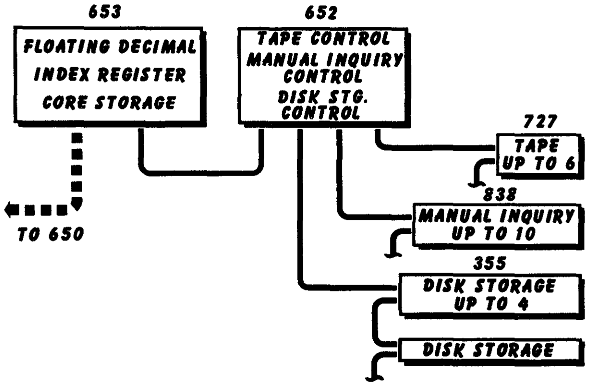


Fig. 3—650 RAMAC (basic 650 not shown).

653 Storage Unit

The next unit in the system is the 653 storage unit, which may contain automatic floating decimal arithmetic, index registers, immediate access (core) storage, or any combination of the three. (See Fig. 3.)

"Immediate access (core) storage" is a necessary part of any system which includes tape or disk storage. The core storage array has a capacity of 60 ten-digit words (plus their signs). It is directly addressable from the 650. Single-word access to transfer to or from the 650 requires the minimum execution time of any 650 instruction, which is two 96-microsecond word time. Block transfers may be made directly between the core storage and the 650 drum in increments of ten and 50 words. Table look-up may be made directly on core storage, the same as on the drum.

In addition to serving as quick access storage for the 650, the 60-word block of core storage serves as a static

† Internatl. Business Machines Corp., Endicott, N. Y.

[1] E. S. Hughes, Jr., "The IBM magnetic drum calculator type 650, engineering and design considerations," *1954 Proc. Western Joint Computer Conference.*

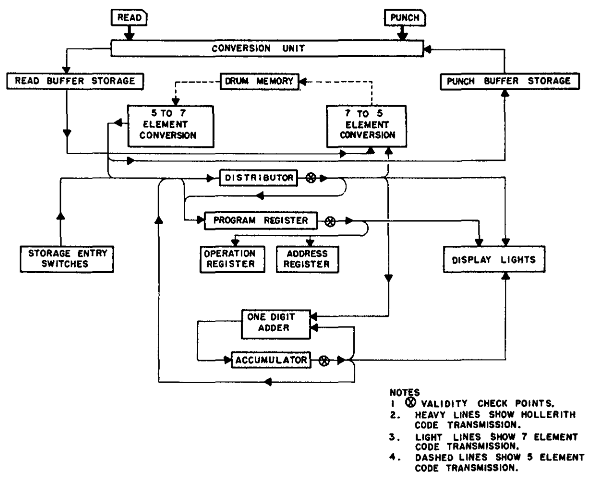

READ
PUNCH
CONVERSION UNIT
READ BUFFER STORAGE
DRUM MEMORY
PUNCH BUFFER STORAGE
5 TO 7 ELEMENT CONVERSION
7 TO 5 ELEMENT CONVERSION
DISTRIBUTOR
PROGRAM REGISTER
STORAGE ENTRY SWITCHES
OPERATION REGISTER
ADDRESS REGISTER
DISPLAY LIGHTS
ONE DIGIT ADDER
ACCUMULATOR
NOTES
1 ⊗ VALIDITY CHECK POINTS.
2. HEAVY LINES SHOW HOLLERITH CODE TRANSMISSION.
3. LIGHT LINES SHOW 7 ELEMENT CODE TRANSMISSION.
4. DASHED LINES SHOW 5 ELEMENT CODE TRANSMISSION.

Fig. 4—Flow of instructions and data in basic 650.

buffer for information transferred between the computer and tape, between computer and disk storage or between tape and disk storage under computer control.

652 Control Unit

The 652 control unit, next in line, Fig. 3, may contain electronic controls for tape; it may contain electronic controls and a thyratron address buffer for disk storage, and it may contain equipment necessary with the manual inquiry feature described in a companion paper.[2] A discussion of tape operation is not included here, except to say that tape units are extremely useful in the application of the 650 RAMAC to in-line processing.

355 Disk Storage

The remaining element in the system is the 355 disk storage unit. (See Fig. 5.) Each disk storage unit has a capacity of six million numeric digits plus six hundred thousand signs, or three million alphabetic and special characters. Information is stored magnetically on both surfaces of each of 50 oxide coated disks. The disks are stacked and rotated on a common vertical axis at 1200 rpm. A description of the prototype disk array and access mechanism has been given.[3] Capacity of each surface is 100 concentric tracks; capacity of each track, as used in the 355, is 600 numeric digits, plus 60 signs, organized into 60, 650-sized, ten-digit words. (See Fig. 6).

There are three independently and simultaneously moveable access arms in each storage unit. (See Fig. 7.) Each arm is forked to straddle a disk and contains a spring-retracted air-extendable read-write head recessed into the end of each "tine" for access to opposite faces of the same disk. (See Fig. 8.)

[2] H. A. Reitfort, "The IBM 650 RAMAC inquiry station operation," this issue, pp. 49–51.

[3] W. E. Dickinson and T. Noyes, "Engineering design of a magnetic-disk-random-access-memory," *1956 Proc. Western Joint Computer Conference*, pp. 42–44.

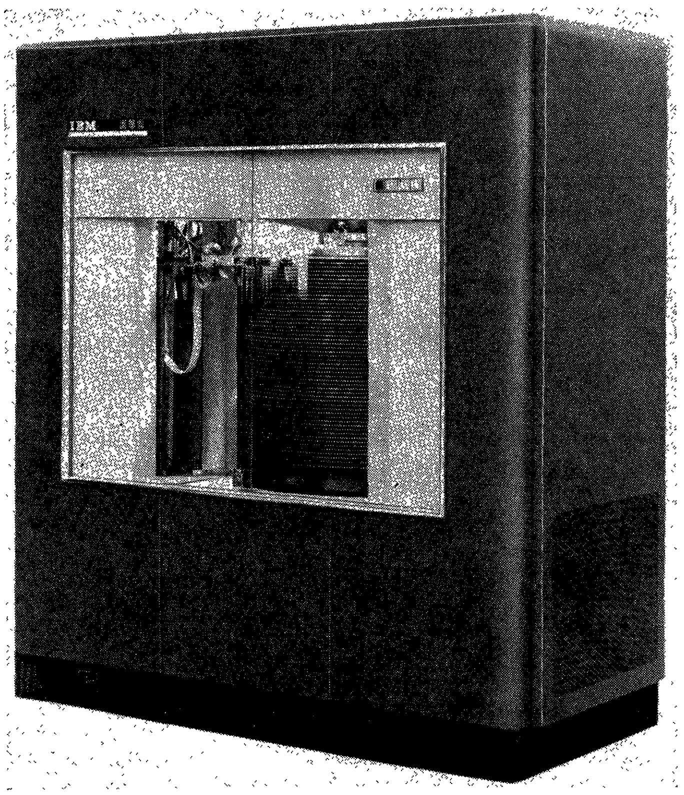


Fig. 5—355 disk storage unit.

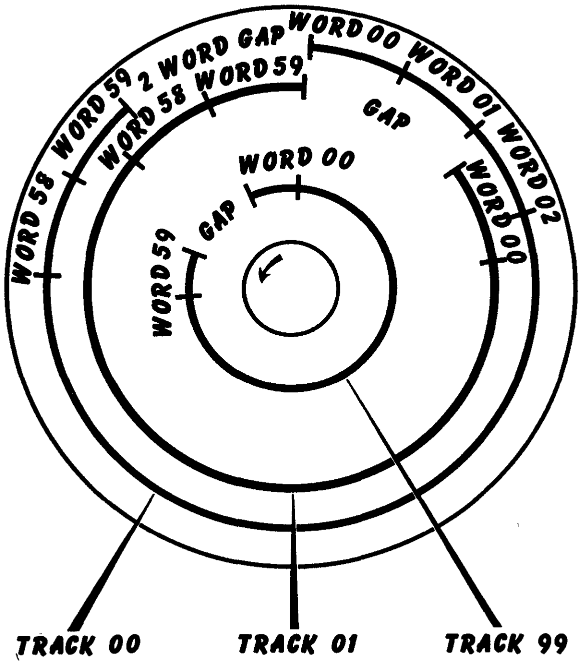


Fig. 6—Track and word arrangement on a disk face.

Disk storage operation is controlled from the computer-stored program by only three commands and a six-digit disk-storage address. The commands are as follows.

OP Code	*Command*
85	seek
86	read
87	write.

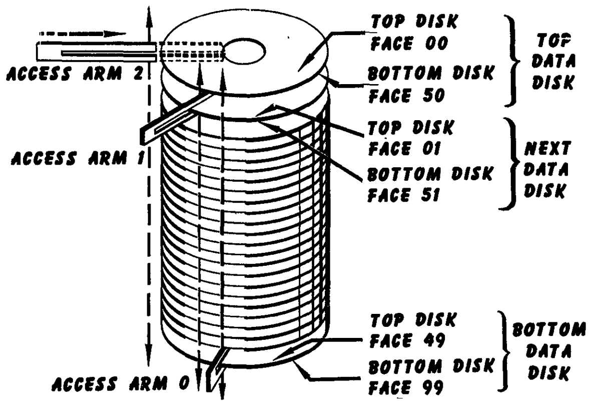


Fig. 7—Arrangement of arms and disks.

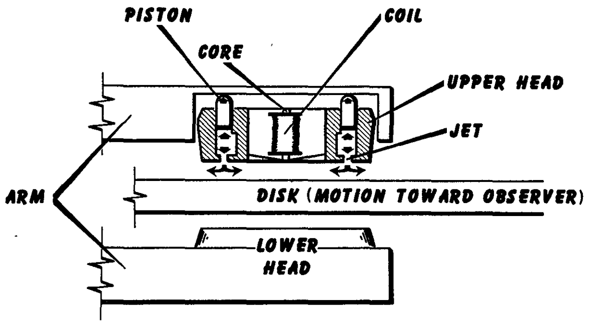


Fig. 8—Air-head arrangement.

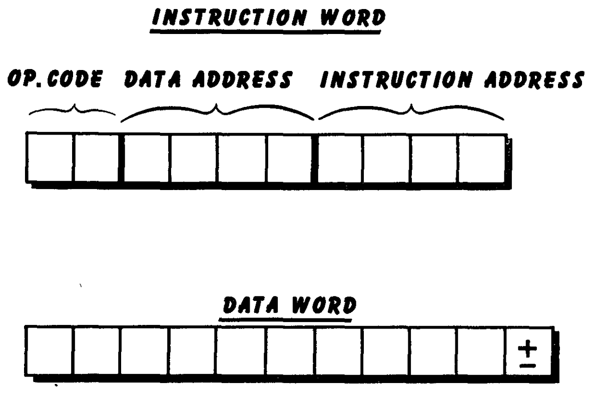


Fig. 9—650 word-arrangement.

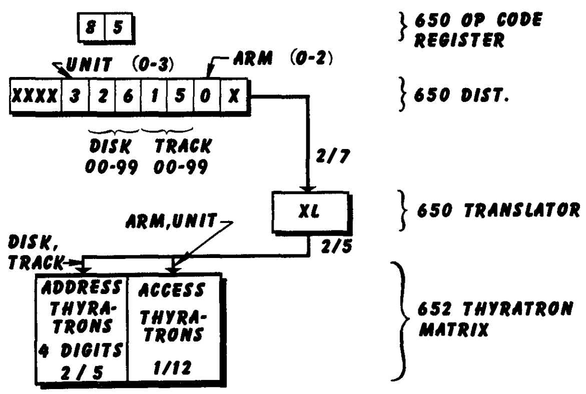


Fig. 10—Seek: transfer disk—memory address from 650 to 652.

In each case the six digit disk-storage address is programmed into the six low-order positions of the 650 distributor, (Figs. 4 and 9), in a program step just preceding any of the three commands. This departure from the usual placement of the data address in the 650 address register is made necessary by the fact that only four places are available in either the instruction word or the address register.

Seek Operation

Seek is one of very few commands issued by the 650 which sets up access to a memory, but which does not make the data transfer in the same step.

At the cost of a few more program steps, the speed of the system is greatly increased by permitting movements of the access arms to be overlapped with each other and with read or write operations in other arms.

Appearance of the 85 in the OP code register sets up seek-mode controls in the 652; thereafter, the seek command is executed in four steps.

1) Transfer of seek address from distributor to thyratron matrix. Information is read out serial by digit, parallel by bit, translated from 2/7 biquinary code to 2/5 code and sent to the 652, where the unit and arm digits are time sampled and cause one of the 12 access thyratrons to fire. The four disk and track digits are time-sampled and stored in a thyratron matrix in a 2/5 code. (See Fig. 10.)

After the two word times required to initiate this much of the seek operation, the 650 finds its next instruction and proceeds with its program.

2) Transfer of the disk and track address from the thyratron matrix to a set of address relays corresponding to the proper arm is made via the contacts of one of the 12 access relays. The address relays are then held through their own contacts. (See Fig. 11, next page.)

A two out of five validity check is made on the address, a bit-for-bit comparison made between thyratrons and address relays, and a one-and-only check made on all the access relays. Satisfaction of these checks extinguishes the 652 thyratron matrix and resets the seek mode controls. The 652 is then ready to accept a read or write command or another seek command. This takes about 30 ms from the initiation of the seek command.

3) Arm servoaction. The information contained in the address relays is translated into a corresponding arm position by action of the servo shown much simplified in Figs. 12,[3] 13,[3] and 14.

Power for both up-down and in-out arm motions is provided by a $\frac{1}{8}$ hp motor. Magnitude and direction of the driving force are controlled by a pair of counter-rotating magnetic clutches. The force is transmitted to the arm by a single steel cable.

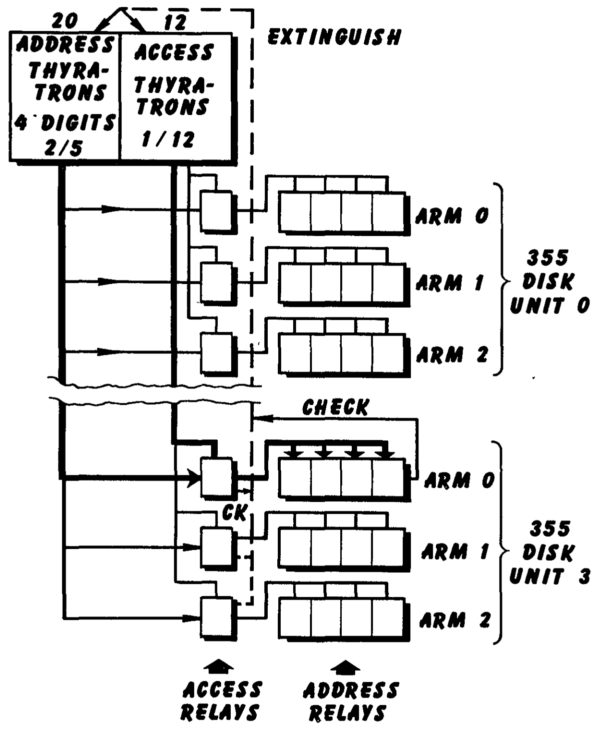


Fig. 11—Seek: Transfer address from 652 to 355 address relays.

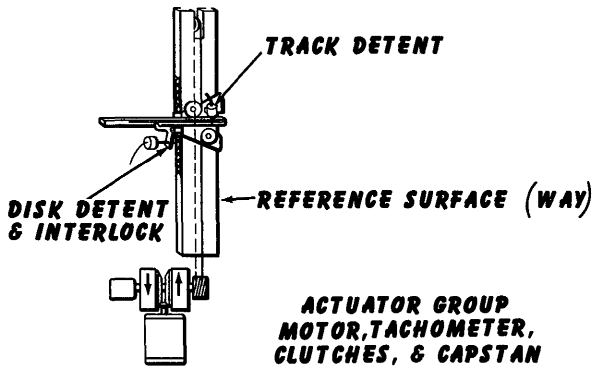


Fig. 12—Mechanical portions of arm-servo.

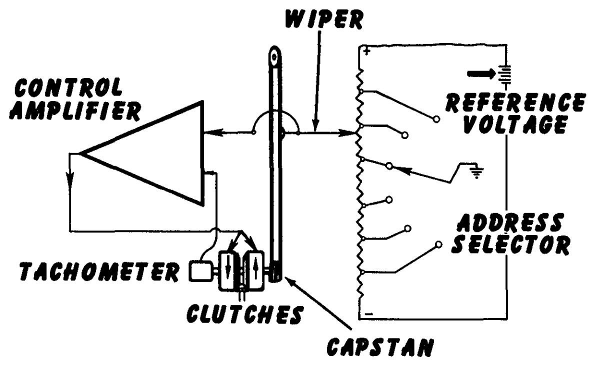


Fig. 13—Electrical portion of arm-servo (simplified).

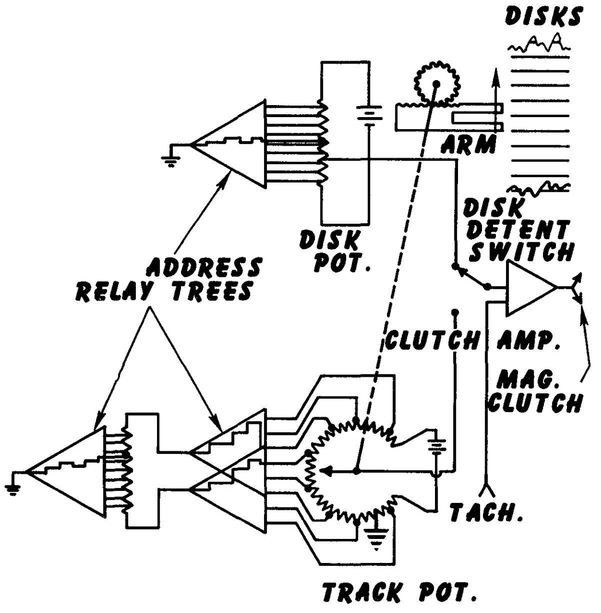


Fig. 14—Electrical portion of arm-servo including disk and track address trees.

When new disk and track addresses are entered into the address relays, new taps are grounded on both disk and track potentiometers through trees of contacts on the address relays; see Fig. 14. The disk error signal causes the arm to be extracted radially clear of the disks, locking the arm in its outermost radial position and unlocking it for vertical motion. The disk wiper error signal then causes the arm to seek and find the new disk null point. The vertical position is then digitalized and locked opposite the proper disk by an air driven "disk detent," unlocking the arm for radial motion and transferring servocontrol to the track potentiometer and wiper. The arm is then moved radially to the track null position. There it is digitalized and locked in place by a "track detent."

4) Final step in the execution of a seek command is to verify that the location of the arm corresponds to the new address.

The arm location is brush-sensed directly in the 2/5 code from a pair of rhodium-plated printed circuit strips. One strip is attached to the top of the arm, the other is attached to the vertical way. A successful bit-for-bit comparison on all four disk and track address digits signals completion of the seek for that arm and prepares it for reading, writing, or another seek.

Failure to get a proper comparison results in movement of the arm to another location, after which it is redirected to the true address and the position check is repeated. Possible random servoerror is thus corrected with a minimum of delay and no special programming.

Total time for completion of a single seek varies, depending on how far the arm is required to move. Minimum time consumed (for seeking from track to track on the same disk) is about 150 ms.

Maximum, for movement from inside track, top disk, to inside track, bottom disk, is about 800 ms. The statistical mean seek time based on random addressing is a little over one-half second.

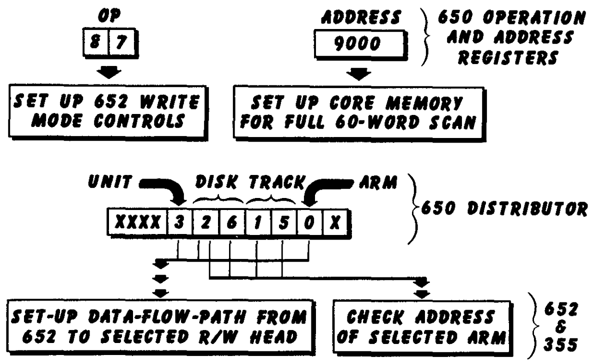

OP
8 7
ADDRESS
9000
650 OPERATION AND ADDRESS REGISTERS
SET UP 652 WRITE MODE CONTROLS
SET UP CORE MEMORY FOR FULL 60-WORD SCAN
UNIT
DISK TRACK
ARM
XXXX 3 2 6 1 5 0 X
650 DISTRIBUTOR
SET-UP DATA-FLOW-PATH FROM 652 TO SELECTED R/W HEAD
CHECK ADDRESS OF SELECTED ARM
652 & 355

Fig. 15—Write flow path setup and address-check.

Write

The function of the write command is to cause the full 60-word content of immediate access (core) storage to be stored on a disk-storage track. The information is written there by a particular read-write head which was placed over the correct track in a preceding seek operation.

Presence of an 87 in the 650 OP code register (Fig. 15), initiates the disk-storage write command, which is also executed in four steps.

1) The six-digit disk-storage address is again transferred from the 650 distributor to the 652 thyratron matrix. The 650 then proceeds with its program.
2) The address information stored in the thyratron matrix is then used for two purposes (Fig. 15): To establish a data flow path from the 652 by means of access relay points to the proper read-write head, and to check that the selected arm is in the proper location. (This is a check on the program to insure that the arm was not inadvertently re-seeked between the intended seek and write instructions for that arm.)
3) Writing begins at whatever point on the track happens to fall under the head at the time the address check is completed. Writing on the track and clocking of core read-out and regeneration are controlled by an 83-kc LC oscillator in the 652.

First, a three-word gap is written, erasing any old information. (See Fig. 16.) Then one digit at a time, core memory is read out and regenerated. Core information is at the same time placed parallel-by-bit into a one-digit buffer, which is scanned out serial-by-digit, serial-by-bit into the read-buss line. (See Fig. 17.)

Information is written on the track in a modified non-return-to-zero form. Presence of a bit is indicated by a transition between the opposite remanent states. (See Fig. 18.)

Writing is terminated upon run-out of the core timing rings. Nominally the last $1\frac{1}{4}$ words of written information overlap the first portion of the written gap. This allows $\pm 1\frac{1}{4}$ word variation (in 63 words) between oscillator frequency and disk speed and at the same time insures the half-word of gap needed by the read circuits to detect the start of a record. It also insures that all of the previous record is erased.

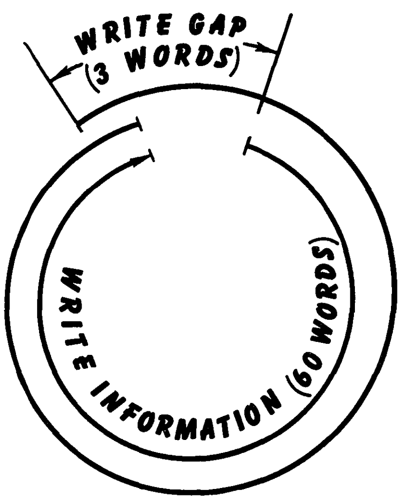


Fig. 16—Track writing overlap.

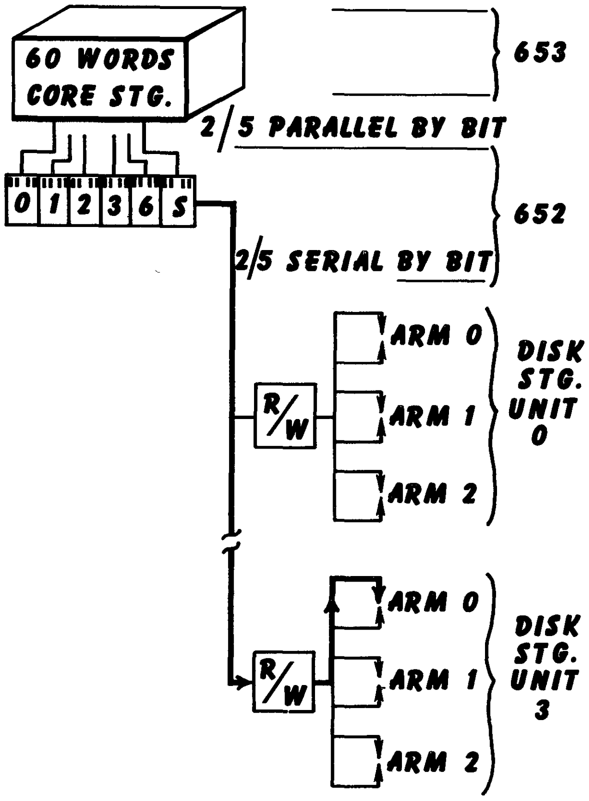


Fig. 17—Write data flow.

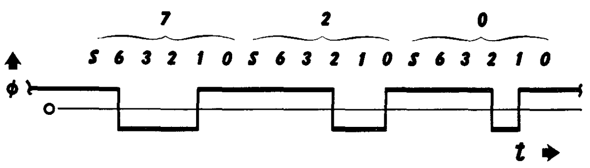


Fig. 18—Modified NRZ recording.

4) To insure that correct, readable information was written, the track is read on the following disk revolution as core memory is again read out and regenerated.

Track information is converted from serial by bit to parallel by bit in the 652, validity checked and compared digit by digit with that from core storage. Satisfaction of this check completes the write operation. Otherwise the operation is repeated from the writing of the gap until the check is satisfied or until the operator intervenes.

Write execution, from initiation by the 650 to completion of the check, requires approximately 30-ms set-up time plus two disk revolutions, a total of 130 ms.

Read

The purpose of the read command is to transfer the 60-word contents of a track into core memory. Execution is very similar to that of the write command.

Flow path set-up and address check are identical.

The beginning of the record is found by sensing the gap. Reading and core timing are clocked by a pair of 83-kc LC oscillators in the 652, (one of which was used for writing). When reading, each bit sensed turns one oscillator off and the other one on, rephasing the clock with the information twice every digit time. Information is scanned serially into a one-digit buffer in the 652 during 6, 3, 2, 1, and 0 bit times, then validity checked and read into core memory parallel by bit during "S" bit time. (Fig. 18.) Satisfaction of the validity check by all digits read signals completion of the read operation. If the check is not satisfied the operation is repeated from the sensing of the gap on. Total time for a read operation varies depending on how long the head must wait for the gap. Average time is about 30 ms for set-up and address check, 25 ms wait for gap, 50 ms for one disk revolution, a total of 105 ms.

Conclusion

Disk-storage operation is controlled by three commands from the computer, seek, read, and write. Average times required are 565 ms for seek, 105 ms for read, and 130 ms for write.

Although access to this disk storage is inherently fast compared to that for other comparable random access memories, speed of the system is materially improved by permitting arm servo actions to be overlapped with each other and with other disk-storage operations.

Independent computer operation is permitted during the execution of any disk-storage command.

All operations are thoroughly self-checked with respect to addressing, valid data transmission, and completion.

Automatic recycling features, which are provided for arm servo, read and write, prevent unnecessary down-time for random errors, without complicating the program.

Discussion

David Zeheb (General Electric): Is the density of recording on a disk variable and if not, does the number of words increase with the distance from the center?

Mr. Royse: The density of recording is variable. It is about a hundred bits to the inch on an individual track and about fifty bits to the inch on an outside track. Because we use just one half of the total radius, the distance is two feet in diameter, and we use practically five inches of the outer area of the disk. We also have sixty words recorded in each track.

N. M. Blachman (Sylvania EDI): Why is the RAMAC clock not recorded on a 51st disk?

Mr. Royse: It was less expensive to do it the way we did.

C. O. Carlson (National Cash Register): What was the reason for three access arms rather than more or less?

Mr. Royse: This seemed to give us an overlap with three access arms, and we could overlap seek operations so that the total seek time, assuming a percentage overlap, was of the order of two-hundred milliseconds.

W. C. Carter (Datamatic): Does the 653 have either floating decimal or index registers or core storage, or some combination of these?

Mr. Royse: You can get any combination. You may have one, two, or three of these features.

G. Barclay (General Electric): Would stationary arms for each disk decrease the access time and make the memory system more economical?

Mr. Royse: They would decrease the access time, but it is our belief that they would increase the cost of the memory system.

C. H. Richards (Convair): What is the function of the "seek" command, if address in information is given when reading and writing?

Mr. Royse: By separating a seek command from a read or write, we are able to send seeks out ahead. We are able to cause seeks ahead of our need for the read or write operation, and because we have multiple arms for each disk, we are able to overlap. We need not wait for the commencing of the seek if we are able to send seeks out ahead.

Mr. Richards: Well then, what function does the address information contain in the read or write? Just a check?

Mr. Royse: It is a check principally on the program and upon the operator. Even in the in-line processing system, there are times when you shut down for maintenance operations. At that point, we lose the information from the address relation in the 355 units and after the maintenance operation is done, we must fire the system up and get valid information back into these address relations before we can proceed. The nature of the system is that it must have valid information back into these address relations in order to get another seek command into the seek operation. Now, to take care of the situation, we have a button on the 652 which is reset, which puts the arm over the zero track on the zero disk. Now, suppose that the system had stopped between the seek which was not an all zero seek, but some other locations in the memory, and the next operation to come up was a read or write command. If we did not have this check on the address, the system might think the arm was correctly in position and we might ruin some valid information in the file. Another reason is that we have to have part of that information in order to prepare the proper data flow pattern between a head and the core memory.

D. L. Shell (General Electric): How long does it take an arm to seek the same track as present location, and how long to seek the same track number on the opposite side of the same disk?

Mr. Royse: We have found that whenever we resend, it takes a little less time than we thought it would originally and this time is in the order of one hundred milliseconds for the complete search of the surface. In other words, in answer to the second part of the question, you have already seeked the track and you have forgotten about it and you try to seek the same location. This takes of the order of fifty milliseconds to accomplish. We must go through all the checking procedures. In seeking the same track number on the opposite side of the same disk is approximately the same time, five milliseconds.

C. F. Summer (RCA Missile Test Project): Would it be possible to address individually any word on any disk? Further, how far apart physically are the channels on each disk?

Mr. Royse: In answer to the first part of the question, no, not directly. The way we handle this is to read an entire track into the core memory and there we have very powerful editing ability. We can make a block transfer of ten or fifty words. From core memory, we can make block transfers in any amount, which includes words which are successive words between core memory and the drum. In addition, we can pull out one word and we can do the reverse. So, what we do to change one word is to read out a track into the core memory, alter the one word, and rewrite the contents of the core memory on the same track. To the last part of your question in regard to how far apart physically are the channels on each disk, they are five thousandths apart.

W. L. Martin (Marchant Research): Does 838 typewriter have a mechanical matrix for automatic typing or are the keys each activated by individual solenoids?

Mr. Reitfort: They are activated by individual solenoids.

A. A. Cohen (Remington Rand UNIVAC): Please expand on how rotation is controlled in servicing waiting inquiry stations.

Mr. Reitfort: As each increase station makes a request, information is stored in relays. Once this station has completed its inquiry and released the typewriter, then we look to see what next station has a request for.

The IBM 650 RAMAC Inquiry Station Operation

HENRY A. REITFORT†

Quick Access Via Inquiry Station Typewriter

A FEATURE of the IBM 650 RAMAC important to "in-line" processing is the facility for quick access to the data processing system from remote locations without interfering with the daily routine. This interrogation of the RAMAC is done through the IBM 838 inquiry station typewriter from locations up to 500 feet from the computer. (See Fig. 1.)

The inquiry station typewriter provides transmission of data to the 650 system and automatic typing of data replies from the system. Up to 10 inquiry stations are available, arranged in one control or in two controls. Each control independently communicates with the 650 system through its own inquiry station synchronizer. Thus, by having two controls, up to twice the volume of inquiries can be handled. By proper programming of the two controls, inquiries and replies can be functioning from both controls at the same time.

Several operating keys and lights are located at each inquiry station which allow the operator to control the various functions of the machine. The unit also contains a regulated power supply and a small relay gate.

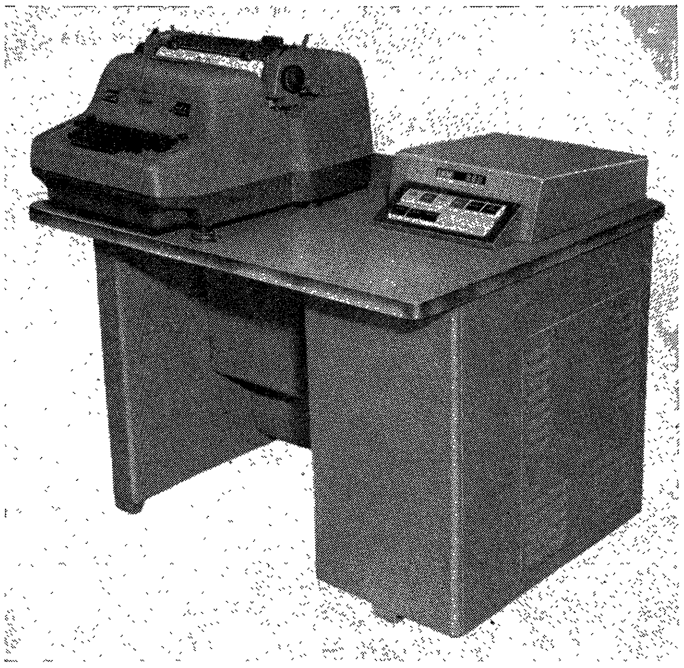


Fig. 1—838 inquiry station.

Flexibility of Inquiry Stations

The inquiry stations are completely flexible since they can inquire into any record in the 650 system. By means of the typewriter keyboard, data and instructions can be provided to the system. Automatic typing of replies to inquiries, miscellaneous messages, or productive output printing can be accomplished.

Each typewriter is addressable from the 650 program, thus an inquiry received from one station can reply at a different station, if desired. A program tape on each inquiry station provides for format arrangement of the inquiry and reply. The program tape also contains a control word that identifies the station and specifies the 650 program routine to be followed for the particular inquiry.

The inquiry stations are connected by multiconduc-

† Internatl. Business Machines Corp., Endicott, N. Y.

tor cables to the IBM 652 control unit, and to each other. Separate transmission channels are provided for inquiries going to the 650 and for replies coming from the 650 to the 838 unit. Thus, while one station is making an inquiry, another station can be simultaneously typing a reply for a previous inquiry.

The control unit contains a small relay gate and an electronic chassis to synchronize the inquiry station with the computer. This unit also contains the 838 checking circuitry. All information from the 838 is buffered with relays in the control unit prior to combining it in the electronic chassis with timing pulses from the computer.

The 652 control unit is an integral part of the 650 RAMAC system. The 650 computer contains the inquiry station input-output synchronizers which are specific bands on the magnetic drum. Each 838 control unit has its own assigned inquiry and reply synchronizers in the computer.

The control of the synchronizers used for inquiries and replies is designed to accept and transmit one character at a time. On an inquiry, the 838 sends one character at a time to the inquiry synchronizer as it is typed by the operator. A reply to an 838 from the 650 consists of transmitting one character at a time from the reply synchronizers to be typed by the 838 at a rate of 600 characters per minute. Each synchronizer contains 100 digits of storage or 10 words of 10 digits each.

Program Tape

Each inquiry station has provisions for a program tape that provides flexibility for 1) forms control through carriage tabulations and spacing, 2) entry and exit arrangement for data transmission, and 3) control and identification of data for 650 processing.

The program tape is a 16-channel perforated plastic tape, the maximum length being four feet long. Each of the 16 channels has an assigned significance. Holes are punched in the tape channels to control the assigned functions. The tape is advanced in conjunction with the programmed movements of the carriage or the control keys on the inquiry station console. Separate program tapes are prepared for each application and are easily interchanged by the operator. (See Fig. 2.)

Each column of the tape is punched to correspond to a given position of the data being sent to the 650 system and all characters being typed in the reply. On an inquiry, as each character is typed, the program tape designates the word and digit position of the input synchronizer where the character is to be transmitted. On a reply from the 650 to the inquiry station, the program tape selects the word and digit position within the reply synchronizer that is to be typed in that location on the paper.

The ends of the program tape are joined together to form a closed loop permitting the inquiry station to proceed automatically from an input format to an output format.

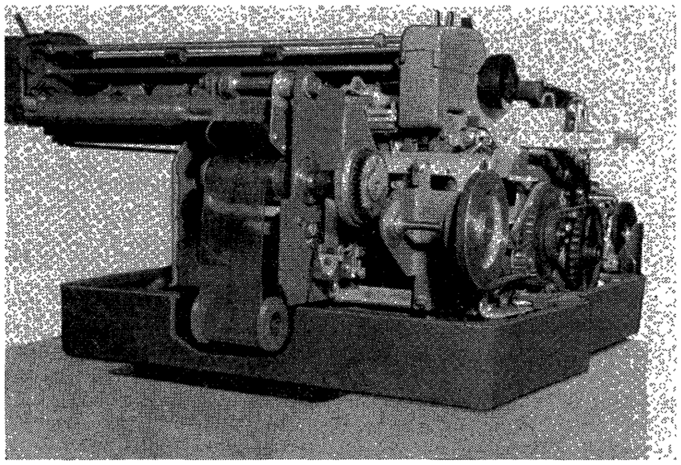


Fig. 2—Rear view of typewriter showing tape lineup.

The 16 channels of the program tape are assigned the following functions:

Channel	Function
1–5	Inquiry synchronizer word location
6–10	Digit position within the synchronizer word
11	Designates an alphabetic character
12	Designates a digit of the control word
13	Start of the input format
14	Start of the output format
15–16	Control carriage functions. Tabulate, carriage return, and space.

Word and Digit Locations

The word and digit locations are punched in a 2-out-of-5-bit selfchecking code. Any word or digit which does not meet the requirements of the 2-out-of-5-bit code is recognized as an error to stop the station during an inquiry. In a reply status, a validity check error prints an asterisk in red in place of the character.

When a hole is punched in the control word channel, the function of the word channels 1–5 are changed from coded word to that of a digit emitter for that tape column for automatic entry of control word into the inquiry synchronizer.

650 RAMAC Communications System

The keyboard communication to the 650 RAMAC system provided by the IBM 838 units begins with the operator requesting permission to send the inquiry or data to the 650. This is accomplished by the operator depressing the request key. The request key checks that the program tape is stopped in the channel designating the start of the input format. If not, the unit automatically advances the tape to the input hole. If the request key is held down when the tape reaches the input hole, it then sends a signal to the 652 control unit asking permission to transmit a message.

When this request is received through the control unit, the 650 erases all information in the inquiry synchronizer from the previous inquiry. It then enters zeros in all positions of the 10 words, and automatically checks that the 10 words of the synchronizer contain zero. If the synchronizer is properly loaded, the proceed light glows at the 838 to inform the operator that the message can be sent. At the same time, the inquiry transmission channels are connected to the inquiry station by relay points. Only one station within a control unit can be connected at one time.

The operator then proceeds to type her message. As each character is transmitted by the 838 and recorded in the particular word and digit position of the synchronizer, as specified by the program tape, it is then automatically read and checked against the character sent by the station. For each character typed, contacts operated by the keys transmit the character information to the relay gate in the control unit. At the same time the synchronizer storage location from the program tape is transmitted to another set of relays. Both sets of relays are checked for validity, and through points of these relays, the information is combined and the character code is transmitted to the inquiry synchronizer and recorded in the proper position on the magnetic drum.

If an error is detected, the entire message must be cancelled by the operator and a new inquiry begun by rerequesting the 650 synchronizer and retyping the message.

When the message is typed and visually verified, the operator depresses a release key to signal the control unit that the inquiry is complete. This can occur only if the program tape has reached the hole in the output channel which is a signal to the system that the necessary number of characters have been typed to satisfy the particular format.

Two operation codes have been added to the 650 system for inquiry stations: a branch-on-inquiry code, and a reply code. The branch-on-inquiry code is located at a convenient place in the normal processing routine. The inquiry station program routines are subroutines that the 650 will branch into if an inquiry is waiting.

The branch-on-inquiry code is a signal to the computer to check the control unit for a released input synchronizer. At this time the ten words of the synchronizer are transferred to the working area of the magnetic drum. The next instruction is the control word entered from the program tape which determines which subroutine is to be followed for processing the inquiry.

Every inquiry should be followed by a reply. Therefore the last instruction of the subroutine should be a reply code. Formulation of data for the reply is accomplished by normal programming The reply instruction transfers the processed data from the working area of the magnetic drum to the reply synchronizer At this time the 650 system returns to the normal processing routine. The reply may consist of 10 words of information or only a single character to indicate the input data was processed.

Determining the Output Format

To determine which station the reply is being sent to, a particular digit of the control word, which has been designated as the station number, is analyzed. This will connect the proper station to the output transmission lines by means of relay points. When the station is selected, the unit checks to make sure that the program tape is at the start of the output format prior to starting the reply. If not, the program tape is automatically advanced until it senses a hole in the output format channel.

The reply is started when the first position of the output format is sensed to determine the location of the first character to be printed. The word and digit position of the reply synchronizer, as designated by the program tape, is analyzed, and by means of relays the proper character is selected to be printed. As each character is printed, the program tape is advanced and the next character location is transmitted to the control unit for selection of the next character. As each character is selected, a validity check of the location relays and the information relays is made. If an error is detected, the ribbon control is operated and an asterisk is printed in red.

The reply continues to be printed until a hole is sensed in the input format channel. This is a signal to the control unit that the reply has been completed and the inquiry station is released. The system is now ready to process the next inquiry and reply.

Installation Components

An installation can consist of many inquiry stations and there will be occasions where several stations may simultaneously request permission to make an inquiry. The control-unit relay circuitry is designed to remember each request and accept only one request at a time. Each request is processed in sequence based upon the station number.

Conclusion

The system as outlined operates with a minimum of 650 RAMAC computing time. The interlocking of the system is such that the inquiry process time through the subroutine is the only time the 650 system is held up.

Two inquiry stations can be operated simultaneously with one control unit, one on inquiry and one in reply. If traffic from the inquiry stations is such that all inquiries cannot be handled fast enough, the second control unit can be added. This will permit two strings of stations allowing four to be used simultaneously, two on inquiry and two on reply.

The punched program tape provides the system with vast flexibility as to the number of formats that can be handled with a minimum amount of computer time since no rearrangement of data is required in the subroutine.

This system, therefore, gives quick access to any or all records in the IBM 650 RAMAC with the required security necessary for combined records. The IBM 838 inquiry station is one more step toward complete facilities for "in-line" data processing.

An RCA High-Performance Tape-Transport System*

S. BAYBICK† AND R. E. MONTIJO, JR.†

Introduction

ONE OF THE severest limitations of data-processing systems today continues to be the low input and output repetition rate capabilities of the electromechanical devices available. While several other forms of external storage have appeared on the market recently, these units suffer in a depth-of-storage and price-per-bit comparison with magnetic tape. Recent advancements in the art of data collection and telemetering techniques are producing a hopeless lag in the data-reduction process with present day computers. The characteristics of available computer-type digital tape transports leave much to be desired in the way of performance when one considers the tremendous amount of data to be stored and processed. The subject of this paper is the development of a specialized digital tape transport intended for computer, data reduction, and special data-storage applications, with the prime purpose of reducing the severity of the problems mentioned.

General Description

This tape transport and its associated equipment is an all semiconductor and magnetics device. Its electronics consist of germanium and silicon diodes, germanium transistors, and magnetic amplifiers. The equipment complement is such as to allow use of the machine with any of six tape widths from $\frac{1}{2}$-inch to $1\frac{1}{8}$-inches wide in $\frac{1}{8}$-inch multiples. Eight to eighteen recording tracks are provided by the series of magnetic heads that accompany the equipment. The main assemblies in the equipment are the tape drive panel, the reel-servo system, the tape-control electronics, the power supply, the local control panel, and the cabinet. Space, air circulation, and power-supply capacity are provided for the read-write electronics which are not included in the package. Type of read-write system and the number of recorded tracks are allowed to remain a function of the application. A front view of the equipment is shown in Fig. 1.

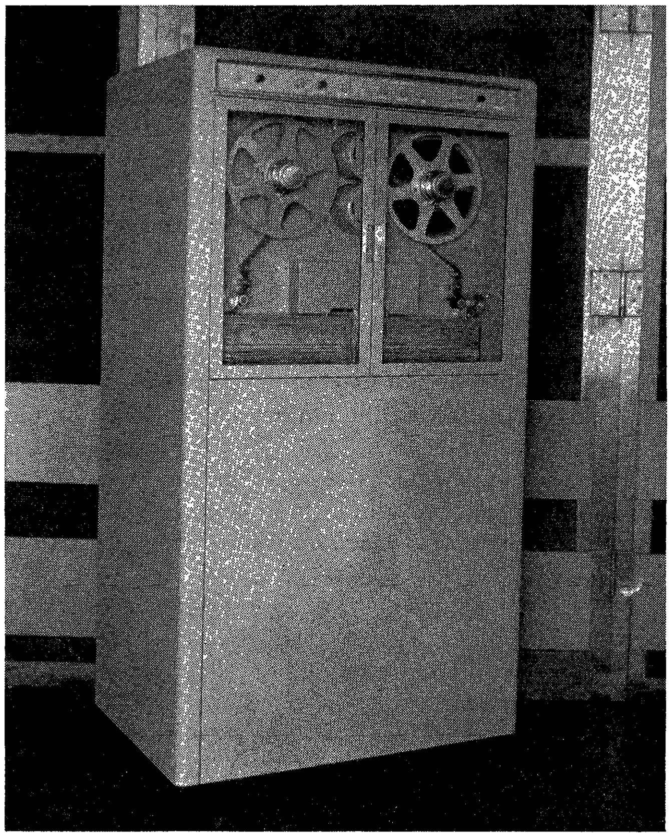


Fig. 1—Tape Transport equipment.

Tape Drive Assembly

The tape drive shown in Fig. 2 (opposite) is a dual capstan device symmetrically designed about a single read-write head. Continuous rotation by the capstan motors in opposite directions provides tape speeds of 100 inches per second and $33\frac{1}{3}$ inches per second in both directions. Two capstan speeds are provided by 12-pole hysteresis synchronous motors with two-speed windings. Magnetic tape buffer storage between the reels and capstans is provided by two small bin assemblies located below each reel. The servo-control system is described by itself in a separate portion of this paper. Rapid start and drive action is provided by jam-roller type clutches for both forward and reverse. Braking is accomplished by two brake shoes which straddle the read-write head. When the tape is stopped, it is clamped between the shoes and the stainless steel surface provided alongside the magnetic head core area. Rough guiding of the tape is provided by two hardened, parallel plates extending between the two capstans. Fine guiding as single-edged referencing is obtained by providing two interference points on the front guide surface to load the tape against the rear guide surface. The tape-drive system belongs to the tensionless-class introduced and used by RCA on the BIZMAC Tapefile. The tape remains in a tensionless condition throughout the tape path, except for two points. Tension is artificially introduced as the tape is unwound and wound on each reel. This action is produced by the reel stripper mechanisms located on each reel and its bin assembly. The reel strippers are

* Work described in this paper was performed under contract with the National Security Agency.
† Commercial Electronic Products, RCA, Camden, N. J.

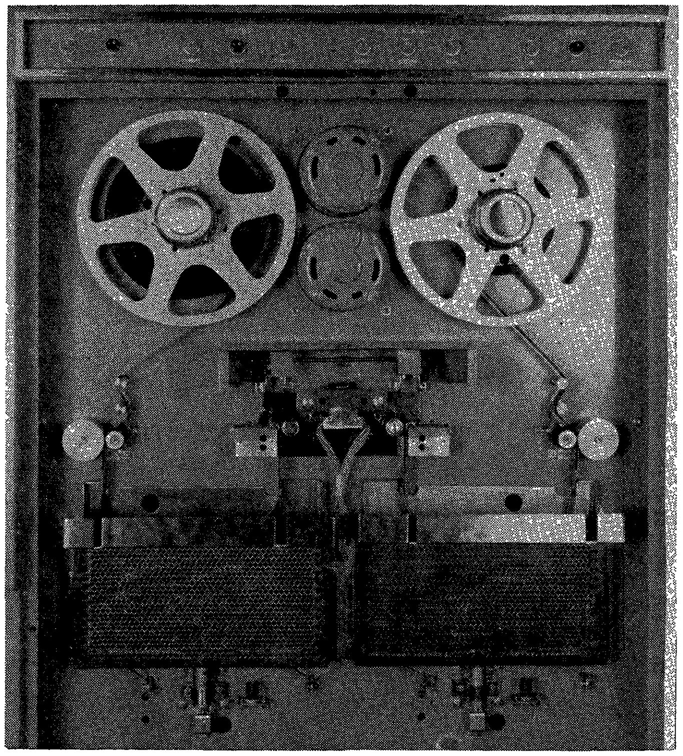


Fig. 2—Tape Transport panel.

driven by reel rotation. When the tape is to be unwound and fed into the bin assembly, the tape is gently pulled from the reel utilizing the reel motion and an overdriven slip clutch in the stripper mechanism. When the tape is wound up on the reel, uniform winding tension is provided by a friction pad that is loaded into position against the tape by reel motion and another slip clutch. The stripper mechanism capstan motion is coupled from the reel shaft through belting which provides a higher peripheral velocity to the capstan than that possessed by the tape as a function of the angular velocity of the reel.

The tape drive is basically designed as a $1\frac{1}{8}$-inch mechanism. Changes in tape width are accommodated by changing or adjusting various subassemblies. Inserts are provided for changing the reel retaining device to accommodate the six widths in three steps. Two guideposts are changed in each reel stripper area for each of the six widths. Changes in the bin assembly are accomplished by providing slots for locating the bin covers and by a set of three curved inserts for the internal tape weighing structure. A separate read-write head and guide assembly is provided for each tape width. This assembly is held by two screws. The entire tape drive can be converted from the narrowest to the widest tape in less than one hour.

Start Mechanism

In order to achieve the fastest start time, the important variables are the mass to be accelerated, the mass associated with the driving mechanism, and the prime mover. Use of the tensionless system enables us to neglect everything else except the mass of the tape which is slight. The mass associated with the driving mechanism consists of a pressure roller and holder (fork) and a simple lever. In order to optimize this design, the pressure roller is kept in motion by means of metal ridges on its periphery as shown in Fig. 3. To reduce further the effect of the roller mass, the fork driving linkage is adjusted so that the pressure roller is only moved 0.003-inch to drive the tape. The same roller, however, is used to drive all widths of tape.

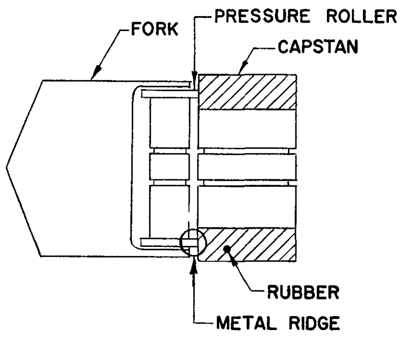


Fig. 3—Plan view of pressure roller and capstan.

Design of the prime mover represents the most difficult problem of all. To achieve fast start time, this device should deliver a large force in the shortest time possible. On the other hand, the instantaneous power demand should be minimized. The actuator that is used is the result of a joint development effort with the speaker development laboratory at RCA. This moving-coil solenoid provides 0.9 pound of force at a stroke of 0.016 inch in 0.5 milliseconds. When this action is converted to start time, it corresponds to starting a $1\frac{1}{8}$-inch tape in less than 2 milliseconds. It was found, however, that a considerable improvement in start time would result from a shaped current pulse with a high starting peak current tapering down exponentially to a steady driving level. When all factors, including actuator life, were considered, a 50-watt instantaneous peak and a 10-watt steady power were settled on to drive the actuator. Use of the shaped pulse decreased the start time to 1.2 milliseconds. Fig. 4 shows the read output of a continuously recorded tape when it is accelerated in this manner. The tape waveform exhibits what appears to be a transient oscillation when the start rate exceeds 10 starts per second. When the tape is subjected to 1000 g accelerations of this type, many peculiar things happen. This is an example of one of them. A combination of various similar effects costs $\frac{1}{2}$ millisecond of start time.

Several series of tests were conducted to determine life and wear of the drive system components and the tape. During the first series of tests, less than 10 million actuations were provided by the actuators because of spring failures. After changing the spring material from aluminum to stainless steel, the next series of tests were stopped after 100 million operations with the actuators

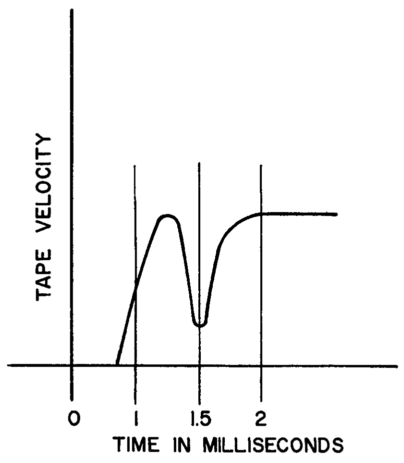


Fig. 4—Output waveform on starting tape.

in excellent condition. Interestingly, tapes were still usable after 100 million start operations at the 120 cps rate.

The Stop Mechanism

The brake actuator is located in the rear of the tape drive panel below the two capstan motors. Its motion is coupled to the brake shoes by a long vertical rod which drives a lever that extends through to the front of the panel. The brake shoes are attached to the lever at a point above the center of the read-write head as shown previously. By braking the tape as close as possible to the read-write head gap line, the brake-shoe assembly acts as a mechanical filter for tape flutter during rapid start-stop operations. The large mass of the brake shoes and brake arms causes the brake shoes to ride on the back of the tape at all times. With very little motion required to brake the tape, very good brake times are achieved. With the same driving power on the brake actuator as was previously described for the drive actuator, a stop time of 1.5 milliseconds is obtained. The first half of this time is required to disengage the drive roller and no deceleration takes place. Linear deceleration follows during the next 0.75 millisecond.

Start-Stop Repetition Frequency

Again, the fundamental problem involved in producing high start-stop repetition rates involved the actuator. The self-resonant frequency of the actuator was 60 cycles, and it was accompanied by several other minor ones in the springs. Through the use of mechanical and electrical damping techniques and very rigid mechanical structures in this area of the mechanism, the start-stop repetition rate was extended beyond 200 start-stops per second.

Tape-Guiding Efficiency

The most important feature in a tape transport is tape-guiding efficiency. This feature also happens to be the most difficult one to check. The guiding characteristics of this tape transport were checked with a very precise rack of read-write electronics whose internal timing error was less than 1 microsecond. Skewness was checked by electronically measuring the time difference between the receipt of the first and last bits in a single character with both bits located at the extreme opposite edges of the tape. This check was made in both directions over the full length of a reel of tape, the width of the tape having been accurately checked on an optical comparator over its entire length. Using this method, skewness was found to be less than $\pm$ 2 minutes of arc when the guides were widest and the tape was at the low end of the specified $\pm$0.0015-inch width tolerance.

Reel-Servo System

The reel servomechanism loop is similar to the time-tested system used on the BIZMAC Tapefile. Control for the reel motors is derived from the weight of tape in the bin buffer storage provided below each reel. Tape weight is converted to electrical information by a differential transformer transducer. The balanced condition for the buffer storage corresponds to the weight of approximately twenty-two feet of magnetic tape. Any variation from null produces a positive or negative output from the differential transformer which in turn is amplified by three stages of transistor amplification. The amplified sine-wave information drives a transistor phase detector where amplitude and phase information is converted to a dc level. Linear preamplification and phase detection both take place on the servo preamplifier plug-in unit, NS-4. The output of each phase detector is amplified by one stage of dc amplification on the phase detector amplifier plug-in NS-9, before the signal is used to drive a differential relay. At this point the servo loses its proportional characteristic and becomes an on-off system. Input and output transfer characteristics for the phase-detector amplifier are shown in Figs. 5 and 6. The circuitry of the NS-9 is designed not only to amplify but also to broaden the dead zone for the loop. The output of the NS-9 is made to approximate a step function so as to produce positive and reliable operation of the differential relay contacts.

One might very logically ask the reason for using a relay at this point. There are several reasons. First, the differential relay is used as a high gain dc amplifier stage. It performs this function with great stability and reliability. It has a power gain of approximately 10,000 and its life is more than adequate. Secondly, functional differences accruing from the use of an on-off loop as opposed to proportional control are very slight. This is attributable to the peculiarities of the error signal derived from the weight sensing system.

The function of the differential relay is to drive the power stage in the servo loop. The power amplifier takes the form of a reversible magnetic amplifier whose 100-watt output drives the reel servo motor directly. The saturable reactor type magnetic amplifier has a power gain of 200.

Stabilization of the servo loop is provided by inexpensive ac rate generators. The output of the rate gen-

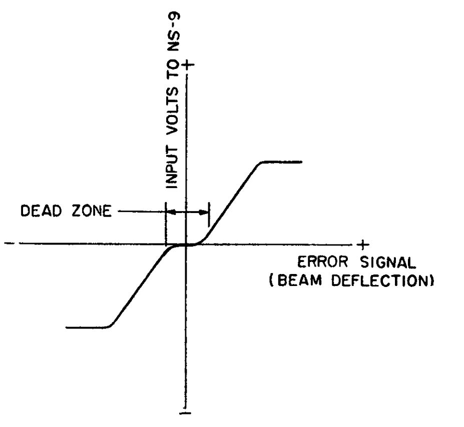


Fig. 5—Input to phase-detector amplifier.

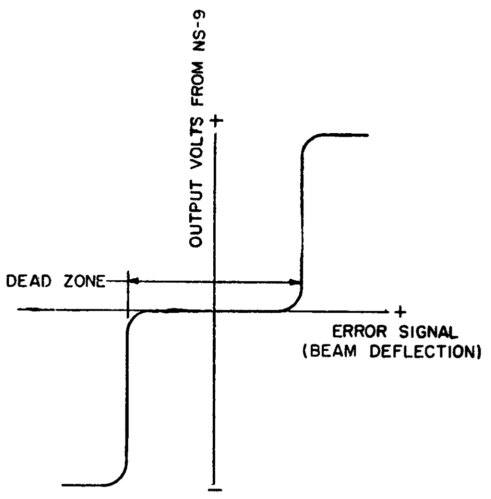


Fig. 6—Output from phase-detector amplifier.

erator varies directly with the surface velocity of the magnetic tape as it passes through the reel stripper mechanism.

Tape-Control Electronics

There are ten types of plug-in units for the tape control electronics shown in Fig. 7. Since all of the circuits in this section employ transistors, the opportunity to provide multicircuit plug-ins is taken. Thus, plug-in units containing two flip-flops, three two-digital gates, three indicator drivers, and seven other multiple circuits are provided. A simplified tape control logic diagram is shown in Fig. 8. Forward, reverse, and stop command inputs accept either pulses or levels. Tape start and stop action is a function of the forward, reverse, and brake solenoids. These high-speed solenoids are driven with a high current waveform as shown in Fig. 9. An impedance of three ohms is presented by the actuator to its driving circuit.

The actuator-driver circuit is a three-stage dc amplifier with an over-all current gain of 1000. This unit is capable of driving a solenoid with 6 amp peaks and 2 amps continuous with generously derated collector dissipation on all components.

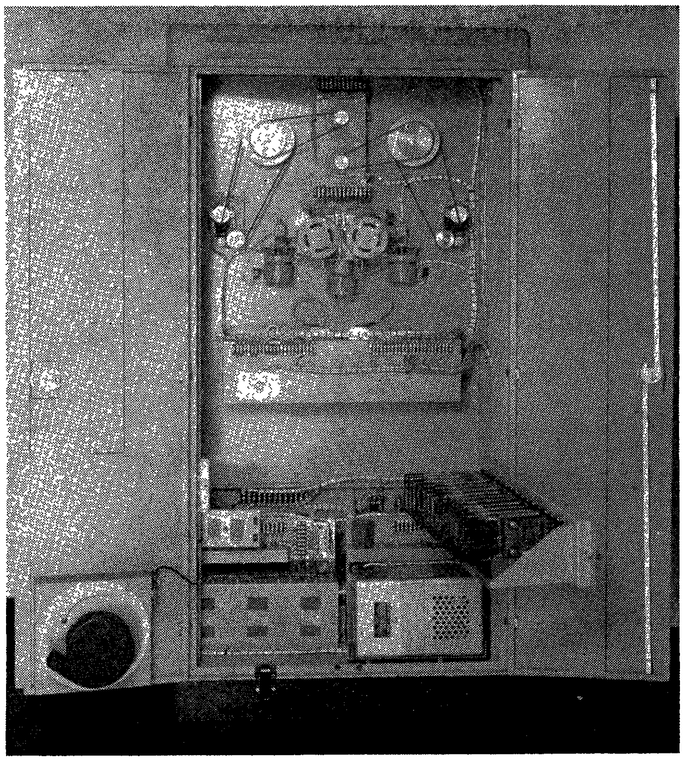


Fig. 7—Rear view of Tape Transport, showing plug-ins.

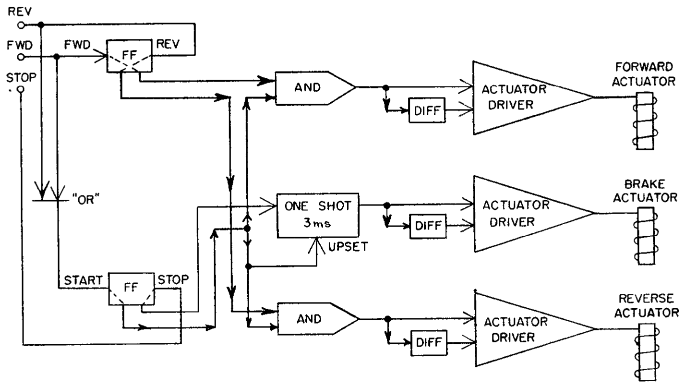


Fig. 8—Simplified tape-control logic.

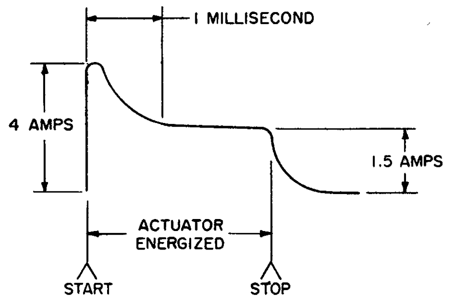


Fig. 9—Input current for high-speed actuators.

Power Supply

The power supply for the tape transport electronics has the difficult requirements of supplying high-current pulses and continuous loads to very low impedance circuits. These requirements result in the need for internal impedances limited to fractions of an ohm in order to provide regulation of less than 5 per cent for load changes of 0 to 200 per cent.

Six dc voltages are provided by the power supply in the range from −48 to +48 volts. Regulation is provided on the input side of the power supply transformers with a constant voltage transformer. Fusing for the supplies is in the transformer primary with sensing relays providing rapid "dc dumping" on the entire supply in the event of the loss of a voltage. The basic power supply circuit is a full-wave rectifier circuit with an inductance-input filter. The rectifying elements are silicon-junction power diodes.

Performance Summary

Before summarizing, the machine characteristics that have been mentioned will be specified and some application data will be noted where it is useful.

Start time: This time varies from 1 millisecond at low start repetition rates on narrow tapes to 2 milliseconds at the maximum repetition rate for the widest tape.

Stop time: This time remains within 1.5 ±10 per cent for all tapes at all rates.

Start-stop space: The interrecord space requirements vary between 0.12 inch and 0.15 inch and is rated at 0.2 inch.

Start-stop rates: The start-stop rate may be varied from 0 to 120 cps for the start-stop specification above. There are no resonances throughout this band.

Reversal time: This time is less than 2 milliseconds.

Skewness: Skewness is specified as less than ± 2 minutes of arc with tape and head interchangeability without adjustment. The timing error produced by this machine allows packing densities of over 750 pulses per inch and character rates in excess of 75 kc on ½-inch tape.

Tape speed variation: Less than 2 per cent, including all effects.

Tape speeds available: 100 and 33⅓ cps in two directions.

Equipment duty cycle: May be operated at highest repetition rate continuously for extended periods.

Remote controls available: Tape-control commands: "forward," "reverse," and "stop."

Conclusion

The equipment described is expected to provide high performance at low cost for general digital data-storage applications. Great flexibility is provided to the user through the wide range of tape widths and the number of information tracks that are available. Reliability has been provided through the use of reliable components and techniques, generous derating practices, and simplified mechanical design. The large number of tracks, high recording density, and extraordinary start-stop characteristics provide powerful tools for using data-storage techniques heretofore considered impractical or too costly.

Discussion

David Zeheb (General Electric): How long would it take an experienced operator to change reels?

Dr. Montijo: A little less than a minute.

C. L. Baker (RAND Corp.): What precautions are taken to prevent oxide build-up on the tape? How many passes can be made on one tape?

Dr. Montijo: Oxide build-up is a very peculiar problem, involving the magnetic head surface, the shape of the surface, the particular type of oxide that is being used; the amount of pressure involved between the tape and the magnetic head. We have designed around this problem by using an all-metal surface. The gap line is fairly sharp so that there is space for trash that comes off the tape surface to fall into. Further, we have specified a hard oxide. The oxide will wear off as a powder and blow away.

Kenneth Olsen (M.I.T.): Is an idler used to keep the tape in contact with the capstan?

Dr. Montijo: The pressure roller, with about 16 pounds behind it, is tapered. This is what is used to drive the tape. At this point the effect of the wrap of the tape on the capstan also provides the force to drive the tape, so that the combination of the two provides enough force to drive the tape. There is no friction, or very little friction, between the tape and these capstans; but since we do have a rubber surface there, the friction between the tape surface and the capstan is rather high, and we do get a gain in the drive as the result of that.

C. F. Summer (Missile Test Project): What pulse packing factors can be used? For a 7-channel head, what is an estimate of cross-talk? Can you write and read on adjacent channels simultaneously?

Dr. Montijo: The cross-talk for the head designed for this machine is down 30 db between the tape and track. The width of the core is 0.032 inch, and the track spacing is 0.063 inch. This is an all-metal surface head, and the packing density is somewhat a function of the head. Practically speaking, it is limited by the quality of the tape surface that is available.

As for the skew problem, the 750 pulses per inch that I mentioned was used with ⅝-inch tape. Instead of the ½-inch tape, we used ⅝-inch tape, because we had a ready supply of it available—some that we used in the Bizmac System, and could test its width accurately. We did not go above 75 kc, because we had a number of circuits in the equipment that would not operate above that point. So that is the reason why we stopped at that particular number.

A Medium-Speed Magnetic Core Memory

GABRIEL E. VALENTY†

Introduction

THE STEADY evolution of the state of the art in coincident current magnetic core memories leads constantly in the direction of an all-transistor system of appreciable proportions. Several organizations have been at work in this field, but the system herein described is the first of its type. It is presented to demonstrate one of several possibilities in achieving an all-transistor memory of 150,000 bits.

The engineering design and the construction of the memory for the Transac S-1000 Computer was contracted to Remington Rand UNIVAC. The computer itself is designed and constructed by the Philco Corporation. The memory requirements are for a 36-bit parallel system with 4096 words of storage with special provisions for reading and restoring specified thirds of any word. Rigorous limitations on weight, volume, and power input were imposed which together constituted an extension in the state of the art when the effort was originally started. Some of the circuit designs will undoubtedly seem awkward in view of the improved transistors now available; however, as computer units, they represented the state of the transistor art and market.

General Description

The logical block diagram shown in Fig. 1 (next page) is introduced here to present a general idea of the entire system. The system is a conventional one using coincident current and inhibit digit control to operate on all 36 bits of the data word in parallel.

The information furnished to the memory by the computer consists of the 12-bit memory address, an initiate signal, the type of operation to be performed (either read, write, or partial write), and the data word or partial data word which is to be written. These signals enter the memory circuitry through buffer amplifiers which convert the computer signals to signals which are compatible with memory circuitry.

Address information is translated in two parts, that which operates the X-coordinate lines and that which operates the Y-coordinate lines. The results of the translation controls drive line switches (current diverters) which provide the coincident current selection of the addressed word.

The initiate signal sets the lock-out flip-flop of the memory timing chain which causes the generation of the first timing pulse. Further stimulation of the timing system is not necessary as additional timing pulses are self-generated. Timing pulses are furnished to appropriate points to govern the sequence of events throughout the memory cycle. The timing of the memory operation does not vary whether a word is being read from memory or a word is being written into memory. The difference between these two operations lies in the control of the gates at the inputs of the memory data register (memory input/output register). To read the memory, the read probe gates (RT_n) are energized; to write into the memory, the write probe (WT_n) gates are energized. These gates are controlled by the $(\text{WRITE})_n$ flip-flops, which store information regarding the type of operation to be performed.

The digit plane control contains all the circuitry necessary to operate 1-bit plane. This includes the sense amplifier, the memory data register and its input gates, and the inhibit/disturb current generator. One-word information transmissions to and from memory are made through the memory data register.

Information furnished to the computer by the memory consists of the word read from memory, a transfer-complete signal which indicates the word is available in the memory data register, and a reference-complete signal which indicates the memory cycle is finished.

Memory Construction

The memory cores are assembled in 38 printed circuit frames, each of which contains 4096 magnetic memory cores. The cores are arranged in a 64×64 configuration (see Fig. 2), p. 59. Two of the core memory frames are reserved as spares.

The magnetic cores are General Ceramics Type S-3 ferrite material which requires a magnetizing force of approximately 350 ma-turns. This type core was selected for its compatibility with transistor current carrying capacity, power dissipation, and voltage and current gain characteristics. Prior to assembly in a core plane, all the cores are individually tested and selected for uniform characteristics. During the test, the cores are subjected to 320 ma full-amplitude current pulses and 190 ma half-amplitude current pulses. When tested with these current pulses, only those cores which met the following specifications were selected for use in the memory: (See Fig. 3)

1) Disturbed "1" output—14 to 18 mv after being subjected to a series of half-amplitude read current pulses.
2) Disturbed "0" output—7 mv maximum after being subjected to a series of half-amplitude write current pulses.
3) Switching time—4.5 ± 0.5 μsec.
4) Peaking time—2.2 ± 0.25 μsec.

† Remington Rand UNIVAC, St. Paul Minn.

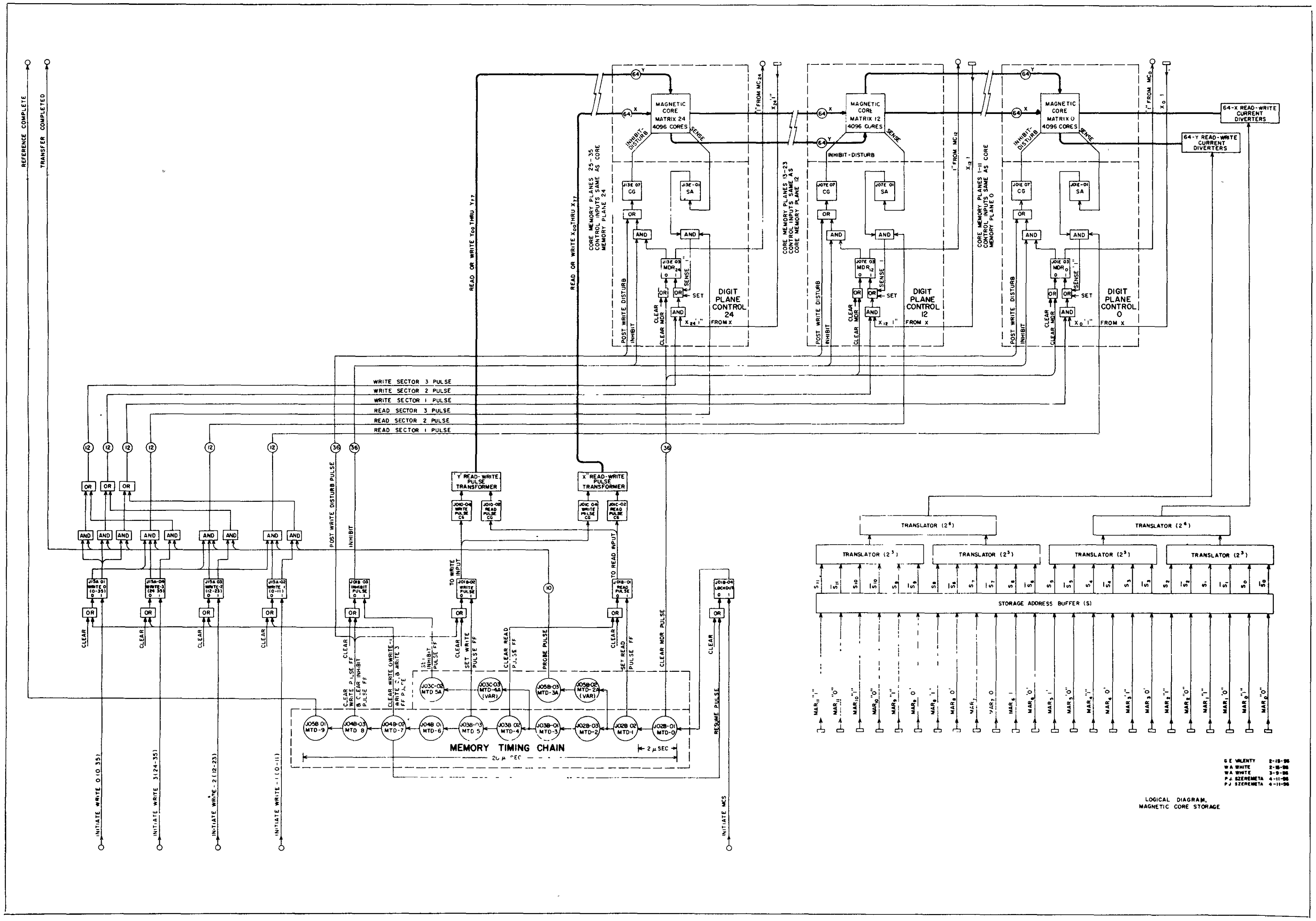

REFERENCE COMPLETE
TRANSFER COMPLETED
MAGNETIC CORE MATRIX 24 4096 CORES
MAGNETIC CORE MATRIX 12 4096 CORES
MAGNETIC CORE MATRIX 0 4096 CORES
CORE MEMORY PLANES 25-35 CONTROL INPUTS SAME AS CORE MEMORY PLANE 24
CORE MEMORY PLANES 13-23 CONTROL INPUTS SAME AS CORE MEMORY PLANE 12
CORE MEMORY PLANES 1-11 CONTROL INPUTS SAME AS CORE MEMORY PLANE 0
DIGIT PLANE CONTROL 24
DIGIT PLANE CONTROL 12
DIGIT PLANE CONTROL 0
64-X READ-WRITE CURRENT DIVERTERS
64-Y READ-WRITE CURRENT DIVERTERS
READ OR WRITE Y00 THRU Y77
READ OR WRITE X00 THRU X77
WRITE SECTOR 3 PULSE
WRITE SECTOR 2 PULSE
WRITE SECTOR 1 PULSE
READ SECTOR 3 PULSE
READ SECTOR 2 PULSE
READ SECTOR 1 PULSE
Y READ-WRITE PULSE TRANSFORMER
X READ-WRITE PULSE TRANSFORMER
TRANSLATOR (2⁶)
TRANSLATOR (2³)
STORAGE ADDRESS BUFFER (S)
MEMORY TIMING CHAIN
20 μ SEC
2 μ SEC
INITIATE WRITE 0 (0-35)
INITIATE WRITE 3 (24-35)
INITIATE WRITE 2 (12-23)
INITIATE WRITE 1 (0-11)
INITIATE MCS
RESUME PULSE
LOGICAL DIAGRAM, MAGNETIC CORE STORAGE

Fig. 1—Logical diagram, magnetic-core storage.

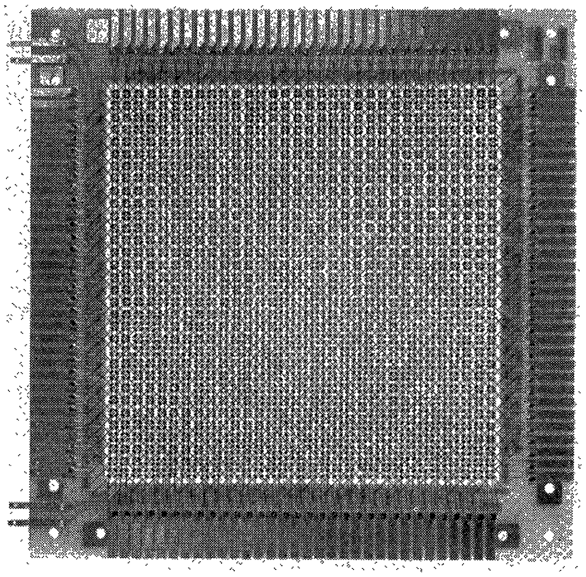


Fig. 2—Magnetic-core matrix.

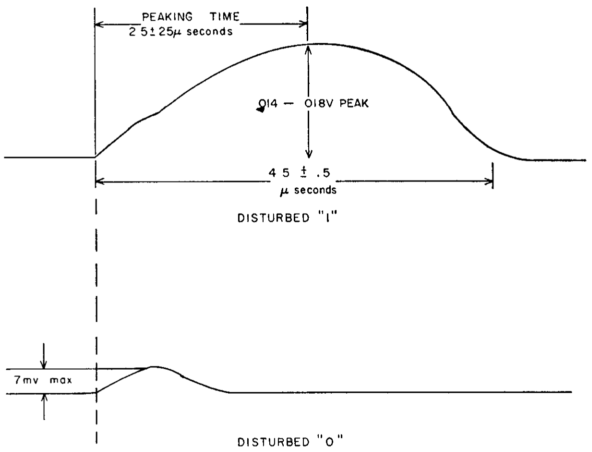


Fig. 3—Memory-core output signals.

The waveforms of the "0" and "1" signals induced on the sense wire are shown in Fig. 3.

Four wires pass through each core: an X-coordinate drive line, a Y-coordinate drive line, an inhibit/disturb drive line, and a sense wire, as shown in Fig. 4.

The sense wire of each core memory plane passes through each core once and is wired diagonally in a symmetrical balanced arrangement so that nearly all unwanted noise signals of all cores except the core being addressed tend to cancel.

The inhibit wire passes through all cores in each core memory plane parallel to one of the coordinate drive lines so that when an inhibit pulse (of opposite polarity to the write pulse) occurs, the magnetic field established on this coordinate line is cancelled.

Each line of the coordinate drive system passes through 64 cores of each plane so that there are 64 X-coordinate drive lines and 64 Y-coordinate drive lines. Therefore, each drive line passes through 2432 cores. The drive lines between planes are connected by a special connector consisting of a number of contacts which

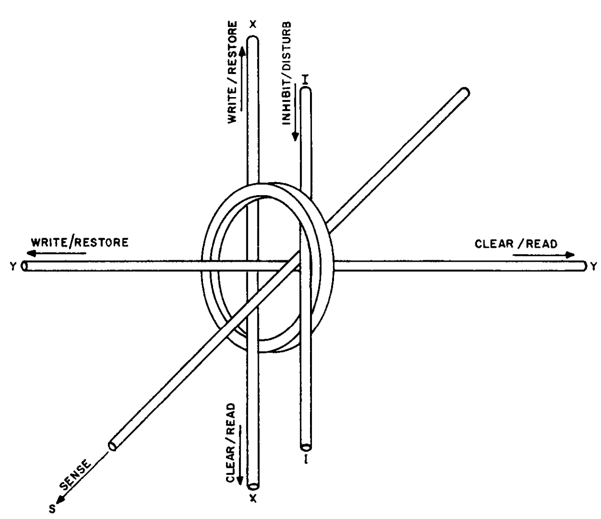


Fig. 4—Magnetic-core control wires. Arrows indicate direction current flow during reading or writing.

resemble cotter keys mounted in a slotted plastic form. This connector is shown in Fig. 5. The cotter key acts as a jumper between the printed circuits of two adjacent core memory planes making the drive lines continuous throughout the length of the memory.

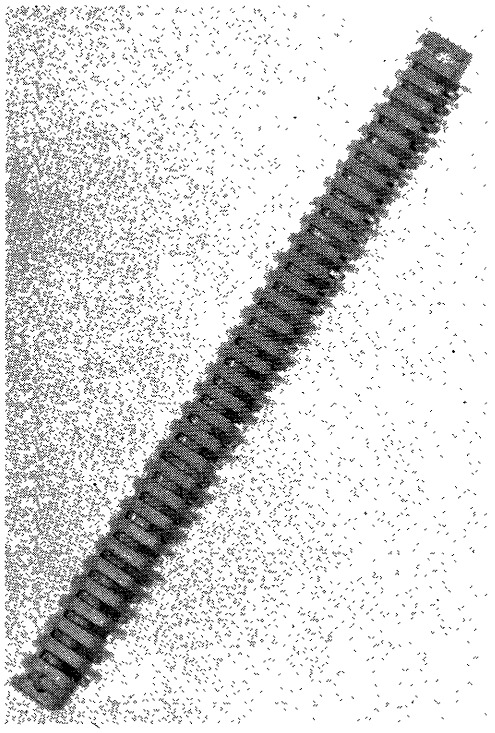


Fig. 5—Memory plane connector.

The memory is mechanically divided into two sections because of space limitations. Each section consists of 19 core memory planes and 2 printed circuit terminal boards. The front terminal board of one section is used to buss together all the drive lines of each coordinate. Single wires from the busses are connected to the outputs of the X and Y read/write current generators. The rear terminal boards are used to connect the two sections, and the front terminal board of the second section connects each drive line to the collector of individual current-diverting transistors.

It has been determined that the magnetic coercive force varies with temperature to such a degree that operation with fixed drive currents over the full range of ambient temperature would not be reliable. To overcome this, the memory is contained in a thermally insulated box and the temperature within the box is maintained constant, slightly above the highest ambient.

Memory Timing

A study of the sequence of timed events which occur during a memory reference, revealed the fact that the requirements for timed pulses and their duration remains fixed regardless of the type of operation and the repetition rate of memory references. This permits the use of a timing device which is not synchronized with computer timing. Therefore, the timing of the memory can be separate from the computer timing, and requires only the receipt of an initiate pulse from the computer. A resume pulse must be supplied to the computer when the reference is completed. This principle is used in the timing system which employs magnetic switch cores to create accurately shaped pulses and transistor switches to set and read the cores. Fig. 6 shows the timing circuit used in the memory.

Transistor Q_1 is biased to cutoff, and a current i_1 flows in the primary winding of the magnetic switch core of sufficient magnitude to hold the applied magnetic field of the core at $+Bs$ (point A) on the hysteresis loop. When transistor Q_1 is turned on by the positive going input pulse, a saturation current, i_2, flows which reverses the applied magnetic field of the core. When this occurs, the magnetic field rapidly traverses the hysteresis loop to $-Bs$ (point B) and sets the core in approximately 1 μsec. A negative going pulse is produced at the output. This pulse is not used. At the termination of the input pulse, transistor Q_1 again cuts off, and read current i_1 begins to flow, producing a positive going pulse in the secondary. This voltage is clamped by diode CR_1. The waveforms shown in Fig. 6 illustrate the voltage relations of the transistor and magnetic core.

The output of the first stage is connected to the input of a second stage so that during the interval that the first core is being read out the second stage transistor is turned on thereby setting its associated core. The subsequent operation of the second stage is like that of the first stage, but it is displaced by one pulse period. Similarly, many stages could be connected in series producing as many pulses (time displaced from each other) as is desired.

The emitter resistor, Re, serves the useful purpose of providing the high input impedance (Beta×Re) necessary to cause the clamping diode to conduct, without seriously reducing reverse base current. In this manner, the response of the transistor (rise and fall time) is made

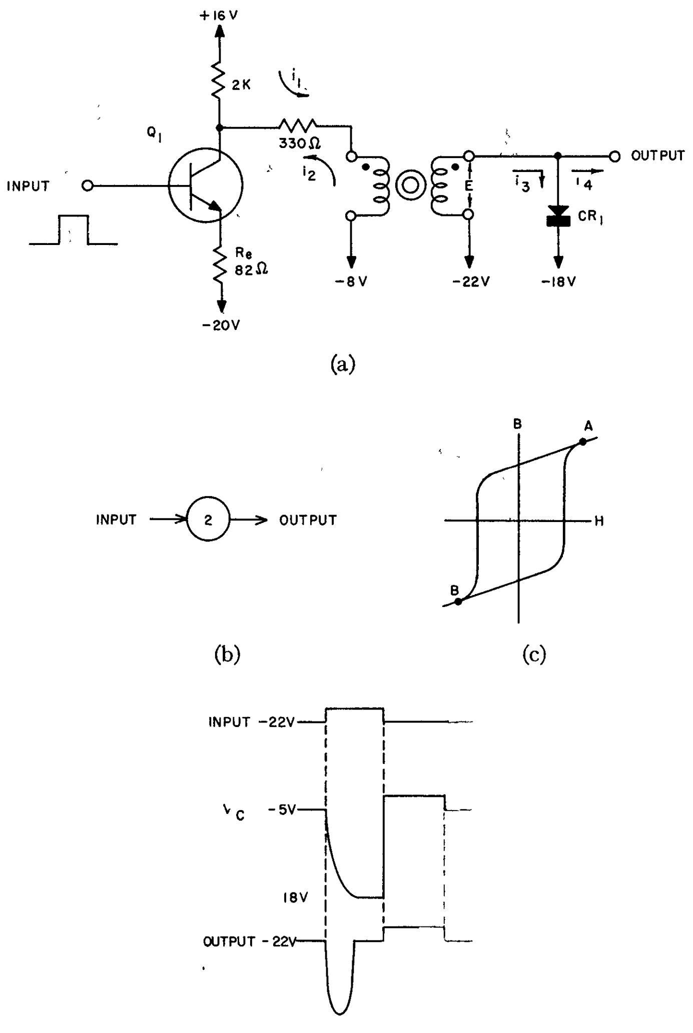


Fig. 6—(a) Magnetic-core timing-delay schematic diagram, (b) magnetic-core timing-delay block diagram, (c) magnetic-core hysteresis loop, (d) magnetic-core timing-delay voltage waveforms.

sufficiently fast so that there is no time lost between pulses of adjacent stages.

The pulse width produced by this method of clamping the output of a magnetic switch core can be accurately controlled so that the pulses produced by many stages are very nearly constant in width. Assuming that magnetic saturation is reached during switching, the core outputs will differ only by that amount of variation caused by differences in saturation flux and by differences in the forward drop of diode CR_1. By precise manufacture and selection of cores, and the use of high conductance diodes with very low forward drop, the variation in pulse width between stages has, in actual circuits, been limited to ± 0.05 μsec. The magnetic switch core used in this application is the Remington Rand Type C core.

It was found that a pulse chain of 10 stages, each producing 2-microsecond pulses, best suited the memory timing requirements. Fig. 7 shows the various pulses required, and Fig. 8 is the simplified block diagram of

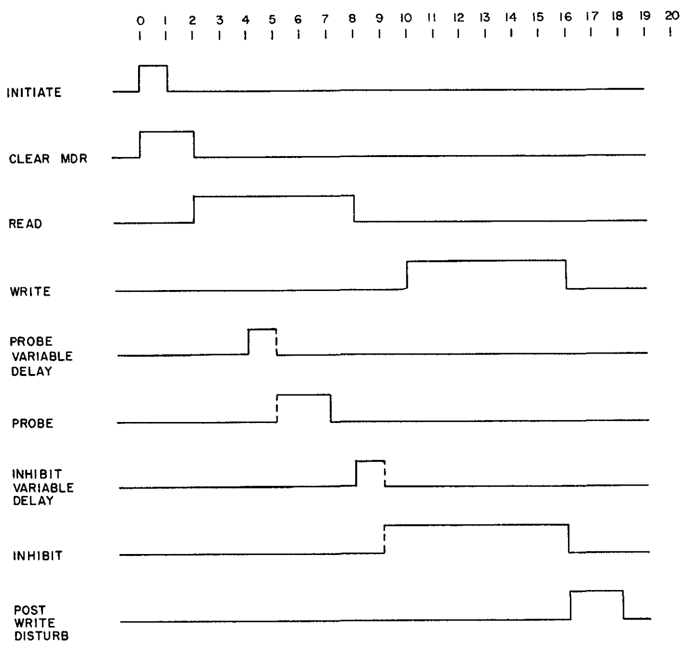


Fig. 7—Memory timed pulses.

in progress. Read, write, and inhibit pulses which initiate the respective current driver circuits are generated in proper pulse width by connecting the set and reset inputs of flip-flops to the appropriate timing pulses. Timed pulses are used directly to clear the memory data register and to generate the probe pulses and the post write disturb pulse. It should be noted that the pulses preceding the probe pulse and the pulse which sets the inhibit flip-flop are variable in width. This is accomplished by making the voltage E, across the secondary winding of the magnetic core, variable. This allows for a manual adjustment of the probe pulse timing and beginning of the inhibit pulse so that they occur at the proper time.

A decided disadvantage of the system is its inherent voltage sensitivity. That is, a change in any one of the three negative voltages, -20, -18, or -22 volts, will cause either a malfunction of transistor Q_1 due to insufficient base current, or, a change in pulse width due to a change in voltage E (shown in Fig. 6). These defects have been overcome by deriving all negative voltages from the same source which is regulated to ± 1 per cent. In addition, a Zener diode voltage reference for the voltage E may be utilized.

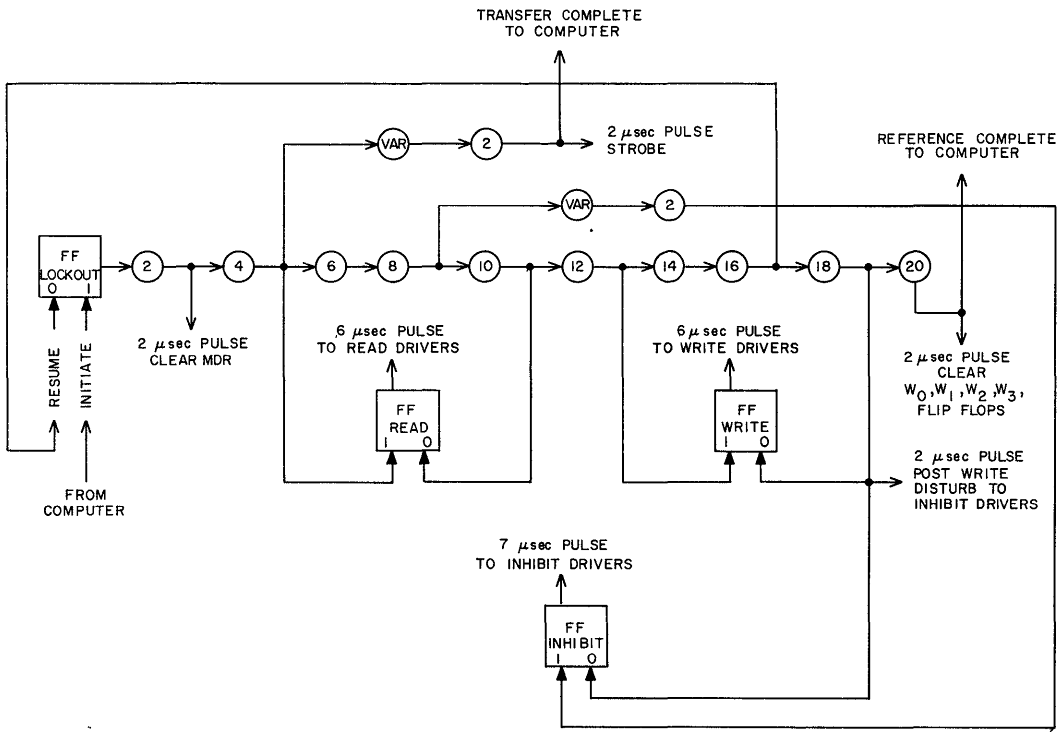

TRANSFER COMPLETE TO COMPUTER
VAR
2
2 μsec PULSE STROBE
REFERENCE COMPLETE TO COMPUTER
FF LOCKOUT 0 1
4
6
8
10
12
14
16
18
20
2 μsec PULSE CLEAR MDR
6 μsec PULSE TO READ DRIVERS
6 μsec PULSE TO WRITE DRIVERS
2 μsec PULSE CLEAR W0, W1, W2, W3, FLIP FLOPS
RESUME
INITIATE
FF READ 1 0
FF WRITE 1 0
2 μsec PULSE POST WRITE DISTURB TO INHIBIT DRIVERS
FROM COMPUTER
7 μsec PULSE TO INHIBIT DRIVERS
FF INHIBIT 1 0

Fig. 8—Simplified block diagram of memory timing chain.

the memory timing chain. A 1-μsec initiate pulse sets the lockout flip-flop which in turn causes the simultaneous readout of the first stage of the chain. To accomplish this, the first stage transistor is normally conducting so that its associated core is normally in the "set" state. The flip-flop output cuts off the transistor and the core is readout. The lockout flip-flop is reset by a resume pulse taken from a stage near the end of the chain so that a reference cannot be initiated while one is already

Logical Decision Elements

The logical circuitry employed to operate the memory consists of simple diode AND and OR circuits.

The logical AND circuit is shown in Fig. 9. In this circuit, the presence of a logical "1" is defined as ground

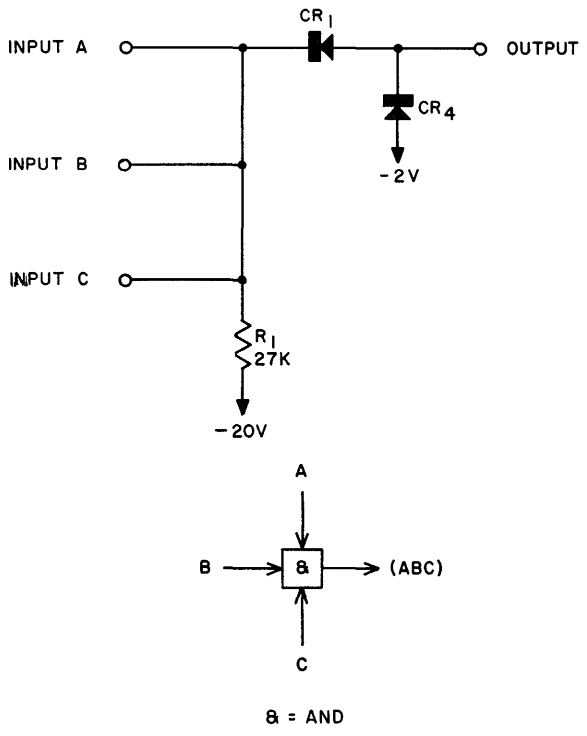


Fig. 9—Basic logical AND circuit and symbol.

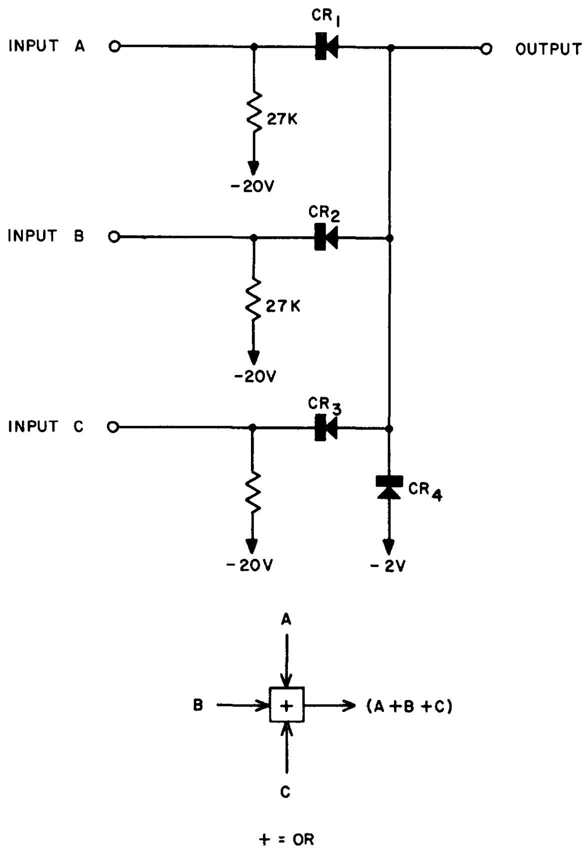


Fig. 10—Basic logical OR circuit and symbol.

potential at an input. When any of the three inputs (A, B, C) are grounded, the voltage at the output will be 0 volts. When all three inputs are open circuited, and only in this condition, the voltage at the output will be -2 volts (clamped by diode CR_4). Thus, the latter state defines the logical AND function.

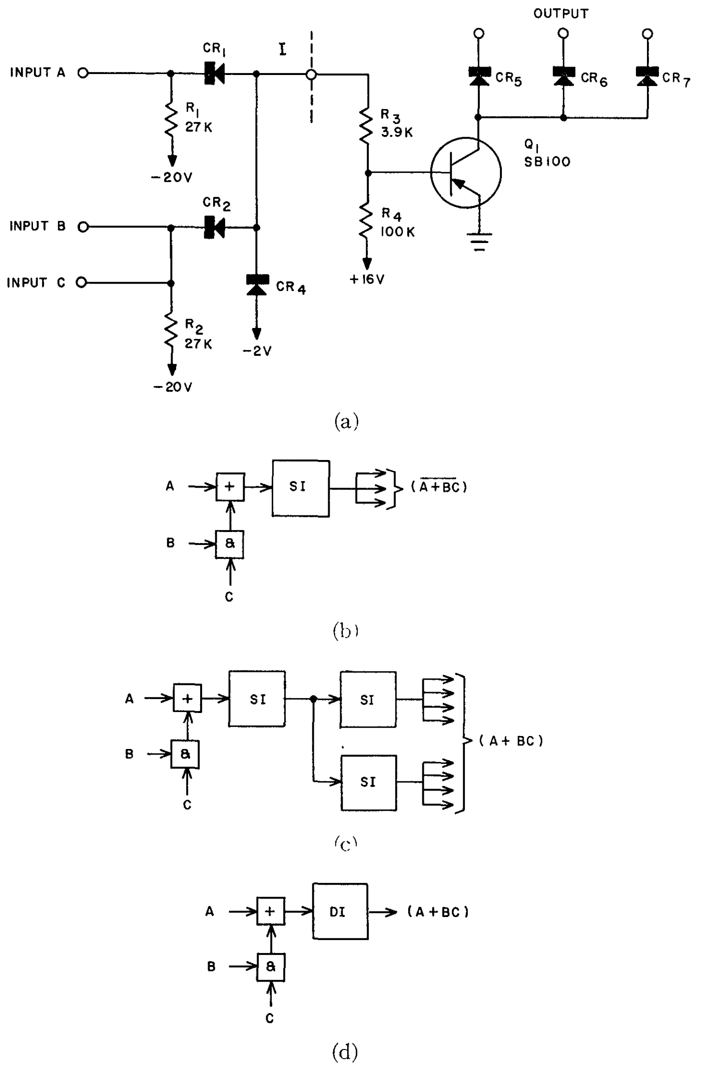
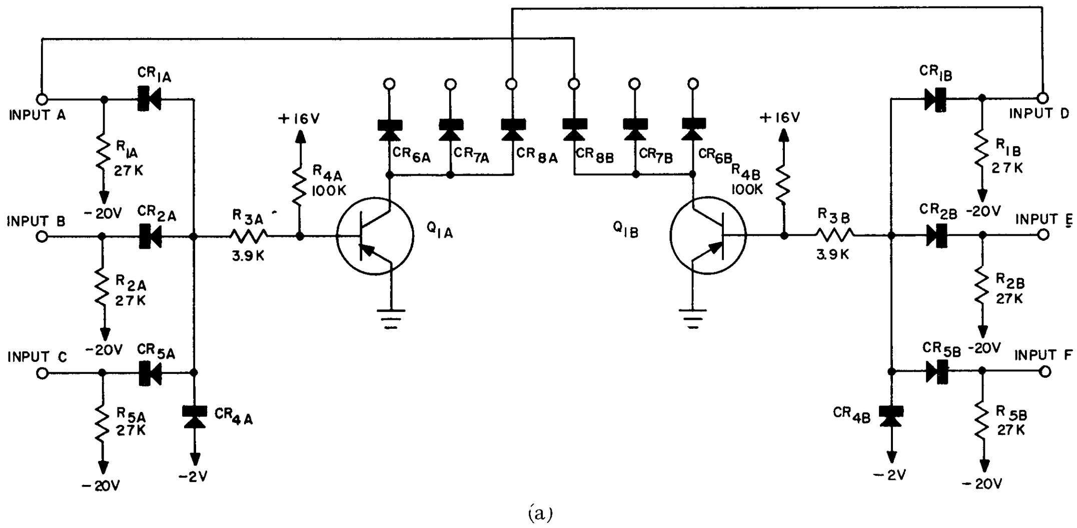
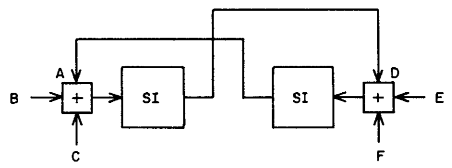


Fig. 11—(a) Basic single inverter circuit, (b) single inverter block diagram, (c) single inverter connected for double inversion, (d) double inverter diagram.

The logical OR circuit (Fig. 10) operates in a similar manner, but diodes CR_1, CR_2, and CR_3 isolate input circuits from each other so that the presence of a logical "1" at any one of the inputs is capable of causing the output to change to -2 volts.

The output of these buffing and gating circuits is connected to a resistive divider network (R_3 and R_4 of Fig. 11) which controls the bias on an SB100 transistor. When the voltage at point I is zero, a small reverse bias is applied to the transistor, holding it in the cutoff state. When the voltage at point I falls to -2 volts, base current is supplied to the transistor and it conducts. Signal inversion in the transistor complements the output of the AND and OR circuits as shown by the accompanying Boolean expression. The output diodes in the collector circuit connect to logical circuits similar to the input circuitry of this stage. These diodes serve to isolate the different logical circuits connected to the same transistor.





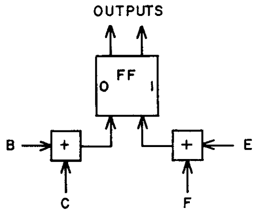


Fig. 12—(a) Basic flip-flop schematic diagram, (b) single inverters connected as flip-flop, (c) flip-flop block diagram.

In order to provide the direct uncomplemented output, it is necessary to connect the output of a single inverter amplifier to a second inverter amplifier. In this case the first stage is used to drive two second stages so as to provide additional outputs.

By connecting the collector of one single inverter to the OR input of a second single inverter, and vice versa, a bistable flip-flop can be constructed as shown in Fig. 12. Outputs from the flip-flop can be taken directly via diodes from the collectors to connect to other logical circuitry.

The three basic circuits: the single inverter, the double inverter, and the flip-flop, provide all the logical control throughout the memory. This includes memory address translation, read and write probe, temporary data storage, and inhibit/disturb logic.

The read or write probe logical signals gate the proper transmission paths connecting to the inputs of the memory data register. The read probe pulse energizes the gates connected to the sense amplifier so that the memory word enters the memory data register. The write probe pulse energizes the gates in the lines from the computer X register so that a new word can enter the memory data register. The memory data register controls the inhibit/disturb gates, thus causing the contents of the memory data register to enter the memory.

There are three different write probe signals; WT_1, WT_2, WT_3 and three different read probe signals RT_1, RT_2, and RT_3. Each of these signals controls the read and write gates of 12 bits of MDR. The RT_1 and WT_1 signals control bits 0–11; RT_2 and WT_2 signals control bits 12–23; and RT_3 and WT_3 signals control bits 24–35. This method of gating permits the reading of the whole word from the memory, writing a whole new word into the memory, or writing new information into a specified sector of the memory.

The read and write probe signals are produced through the use of the logical circuits shown in Fig. 1. Inverse logic is employed to create the read probe signals. One of the four flip-flops shown may be set by a signal from the computer at the same time that a memory reference is initiated. Flip-flop W_0 is set if an entire new word is to be written in the memory. Flip-flop W_1 is set if new information is to be written in bits 0–11; flip-flop W_2 is set for bits 12–23, and flip-flop W_3 is set for bits 24–35. The flip-flops are cleared at the end of the reference by a pulse from the timing chain. The proper outputs of the W_0 flip-flop are combined with the proper outputs of the W_1, W_2, and/or W_3 flip-flop outputs into the correct logical gates. The results of these combinations are combined with the probe pulse from the timing chain.

Therefore, if none of the flip-flops are set, all three of the read probe signals will be present allowing the whole word from the memory to enter the memory data register. If the W_0 flip-flop is set, all three of the write probe signals will be present allowing the whole computer word to enter the memory data register. If the W_1

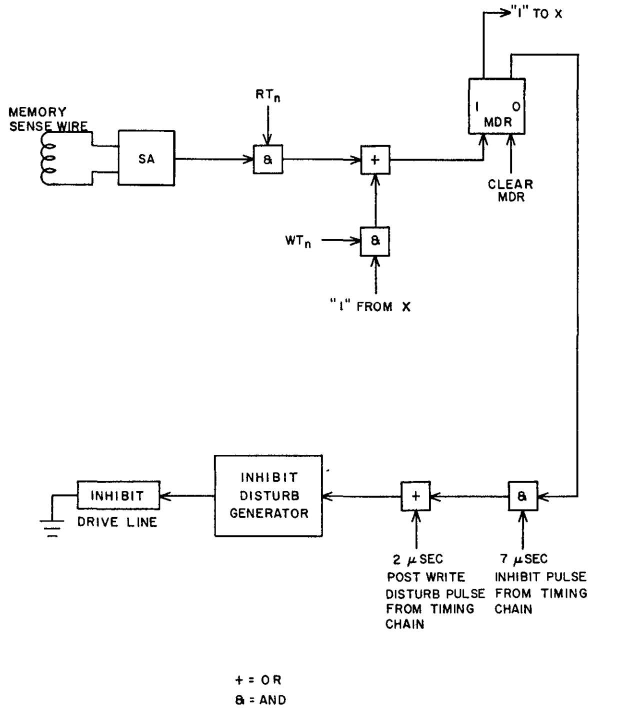

MEMORY SENSE WIRE
SA
RT_n
&
+
"I" TO X
I O MDR
CLEAR MDR
WT_n
&
"I" FROM X
INHIBIT
DRIVE LINE
INHIBIT DISTURB GENERATOR
+
&
2 μSEC POST WRITE DISTURB PULSE FROM TIMING CHAIN
7 μSEC INHIBIT PULSE FROM TIMING CHAIN
+ = OR
& = AND

Fig. 13—Logical block diagram of restoration circuit.

flip-flop is set, the WT_1, RT_2, and RT_3 pulses will be present, allowing bits 0–11 of the X register and bits 12–35 of the memory to enter the memory data register, etc. The other read and write functions are formed similarly. Subsequently, the contents of the memory data register are written into the memory.

The manner in which the memory data register controls the inhibit/disturb generators, which, in turn, controls the writing of "1's" or "0's" into the memory, is shown in Fig. 13. The memory data register is cleared by a 2-μsec pulse at the beginning of the memory reference in preparation for receipt of the new word from memory, or the X register, or a combination of both. The output of the sense amplifier and the read probe gate is shown connected to an OR circuit with the output from the X register and the write probe gate. The "1" output of the memory data register is connected to the X register while the "0" output side is connected to an AND gate with the 7-μsec inhibit pulse from the inhibit flip-flops. The output of this gate is connected to an OR circuit with the 2-μsec post write disturb pulse from the timing chain. This line controls the operation of the inhibit/disturb generator. Thus, for those bits in MDR containing "1's," a 2-μsec post write disturb current pulse will appear on the appropriate inhibit/disturb lines, and a "1" will be written into memory. For those bits containing "0's," however, a 9-μsec inhibit current pulse will appear on the appropriate inhibit/disturb lines and a "0" will be written into memory. For the proper timing of these two pulses refer to Fig. 7, current waveforms.

Address Translation and Line Selection

The 64×64 memory core configuration requires 64 X-coordinate drive lines, and 64 Y-coordinate drive lines.

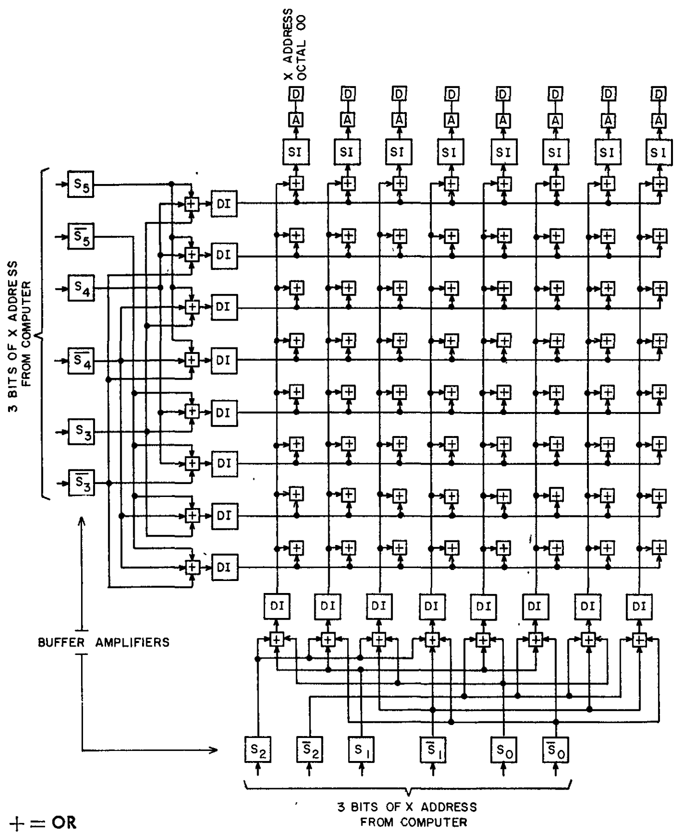


Fig. 14—Simplified block diagram of translator.

All the X-drive lines are bussed together at one end and connected to the secondary of the read/write transformer. The other end of each line is connected to the collector of a bilateral transistor which is turned on by the address translator when selected. The Y-coordinate drive lines are connected in the same manner.

Twenty-four buffer amplifier stages are used to make the signals from address transmission lines from the computer (Philco circuitry) compatible with the memory logical circuitry. These are simple isolation transistors in series with the signal lines. The remainder of the address translation (Fig. 14) is quite conventional with the exception that inverse logic is employed to reduce the number of standard logical circuits employed. The address buffers feed 16 three-input "OR" circuits to operate the appropriate double inverter amplifiers (DI) performing an octal translation for the X line selection. (The Y line selection is made by an identical system.) The outputs of the double inverters are fed to 64 two-input "OR" circuits completing the translation. This is followed by single inverter amplifiers (SI) which complement the input to restore the proper sense.

The output of the single inversion is connected to the diverter amplifier (A) and then to the line selecting current diverter (D). Fig. 15 is the circuit of the last three stages of the translator. The primary purpose of this circuit is to change the current level from that of the surface barrier transistor to that of the current diverter (200 ma). Transistor Q_1 supplies approximately 1 ma to the base of transistor Q_2 keeping it in the saturated conductive state. A voltage of approximately -2 volts

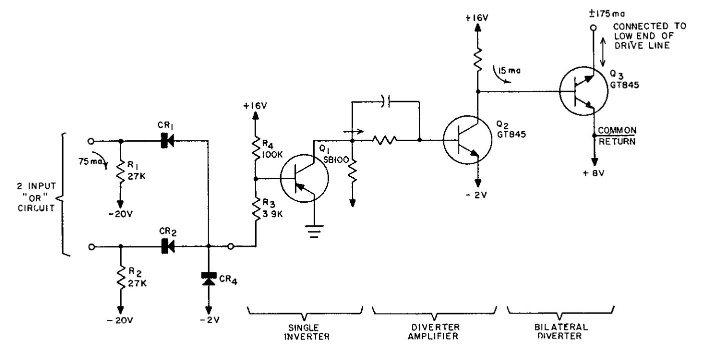


Fig. 15—Current diverter schematic diagram.

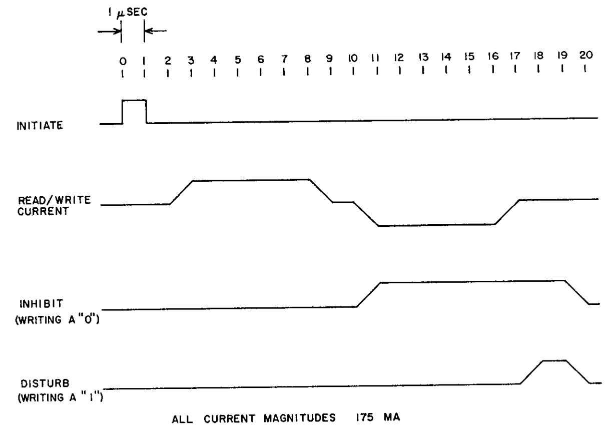


Fig. 16—Current generator waveforms.

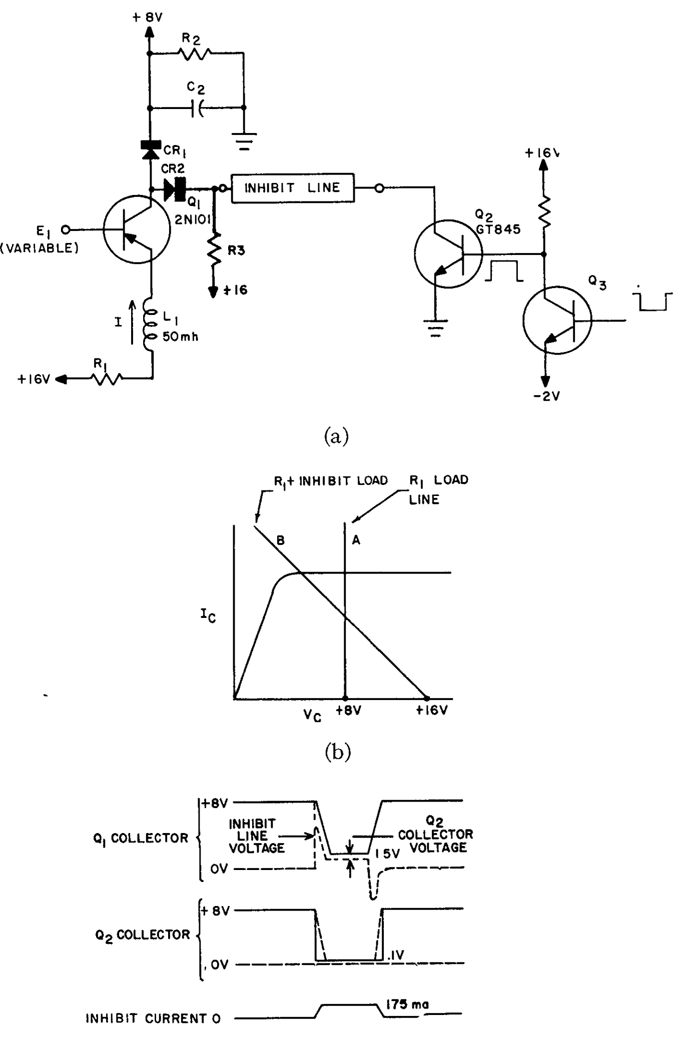


Fig. 17—(a) Inhibit/disturb current generator schematic diagram' (b) transistor Q, collector characteristics, (c) inhibit/disturb current generator voltage waveforms.

appears at the collector of transistor Q_2 and also at the base of transistor Q_3. The diverter transistor of only one drive line conducts while all other lines remain cut off. Therefore, the voltage impressed on the collectors of all cutoff transistors will reach the peak voltage produced by the current generators. It is necessary to reverse bias the cutoff transistors at a voltage greater than the expected collector voltage variations. Thus, in this system, Q_3 is cut off with 10 volts reverse bias.

When a drive line is selected, transistor Q_1 and Q_2 cut off, and a 15-ma base current is supplied to transistor Q_3. Transistor Q_3 is designed with equal forward and reverse gains so that it can easily pass bipolar pulses of equal magnitudes. The response time of transistor Q_3 in this circuit is such that a 200-ma pulse can flow 2 μsec after the base current begins to flow and will cut off 2 μsec after base current ceases to flow.

Constant-Current Generators

The current waveforms required to operate the memory are shown in Fig. 16. The inhibit/disturb current generator produces either of the two pulses shown. The inhibit pulse is a 9-μsec pulse which overlaps the write current pulse and is of opposite polarity to it. The occurrence of this pulse causes a "0" to be written into the addressed memory core. The disturb pulse is a 2-μsec pulse occurring immediately after the write pulse whenever a "1" is written into the addressed core. This places the core in the disturbed "1" state immediately so that the half-disturb noise signal is greatly reduced during subsequent memory references involving either of the two coordinate drive lines which pass through this core. This results in greater uniformity of 1-signal output from all cores.

Several factors were taken into account in the design of the constant-current generators; regulation of current magnitude for variation in load or supply voltage, stability of current magnitude regardless of transistor gain, ease of varying current magnitude, and the transient response of the power supply.

Fig. 17 represents the circuit employed as the inhibit/disturb current generator. The low-frequency transistor (Q_1) is biased by base voltage, E_1, to conduct at the desired inhibit current magnitude. The high-frequency response transistor, Q_2 (Type GT845), is normally cut off and therefore the dc current flowing in transistor Q_1 passes into the resistance R_2. The transistor Q_2 load line (exaggerated) is shown as line A on the graph of Fig. 17. When transistor Q_2 is switched into saturation by the proper timed pulse, the collector voltage of transistor Q_1 drops rapidly below +8 volts so that diode CR_1 is cut off. Consequently all the pre-established dc current

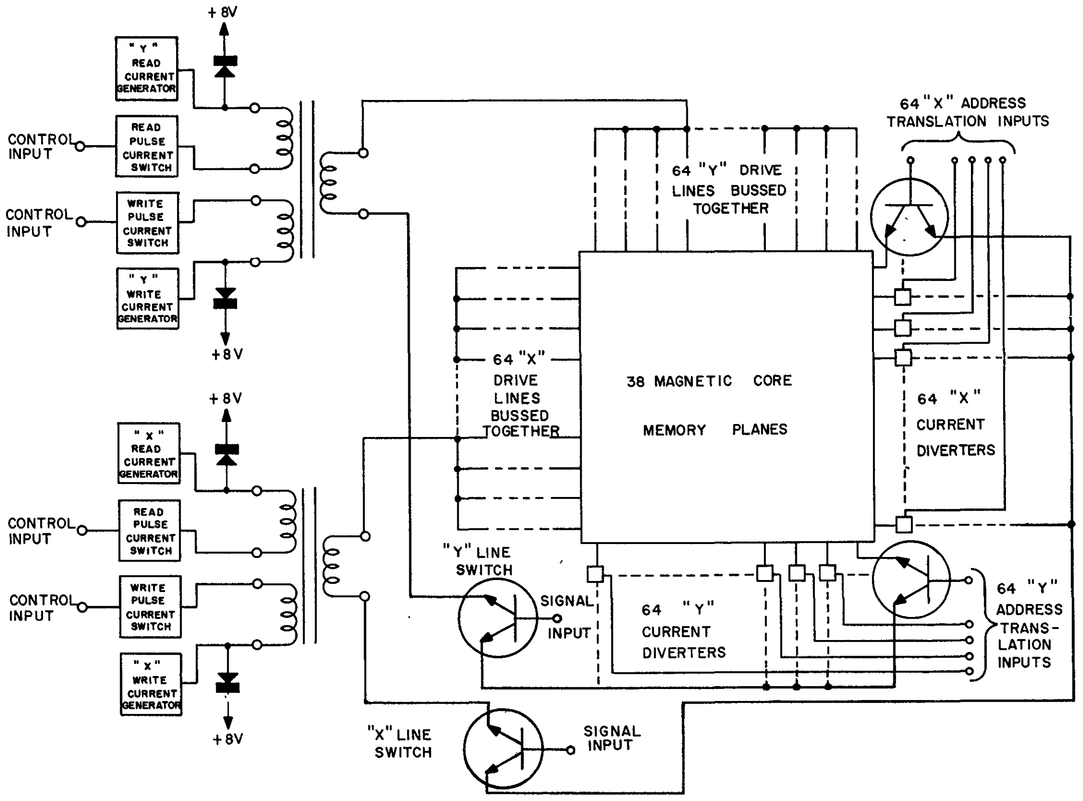

+8V
"Y" READ CURRENT GENERATOR
CONTROL INPUT
READ PULSE CURRENT SWITCH
CONTROL INPUT
WRITE PULSE CURRENT SWITCH
"Y" WRITE CURRENT GENERATOR
+8V
+8V
"X" READ CURRENT GENERATOR
CONTROL INPUT
READ PULSE CURRENT SWITCH
CONTROL INPUT
WRITE PULSE CURRENT SWITCH
"X" WRITE CURRENT GENERATOR
+8V
64 "Y" DRIVE LINES BUSSED TOGETHER
64 "X" ADDRESS TRANSLATION INPUTS
64 "X" DRIVE LINES BUSSED TOGETHER
38 MAGNETIC CORE MEMORY PLANES
64 "X" CURRENT DIVERTERS
"Y" LINE SWITCH
SIGNAL INPUT
64 "Y" CURRENT DIVERTERS
64 "Y" ADDRESS TRANSLATION INPUTS
"X" LINE SWITCH
SIGNAL INPUT

Fig. 18—Simplified diagram of "X" and "Y" drive line and current diverter system.

of transistor Q_1 now flows through the inhibit/disturb line in the form of a current pulse. Load line B is representative of the condition during the flat portion of the current waveform.

During the transition period, the collector capacity charge variation tends to increase the collector current which causes an overshoot followed by a drooping waveform due to the slow response of transistor Q_2. The inductance, L_1, is inserted in the emitter circuit to provide degenerative feedback and overcome this undesirable effect. This keeps the collector current constant during the transition periods.

Diode CR_2 in series with the inhibit/disturb drive line and the resistor R_3 connected to +16 volts serve to reduce the capacitive loading of the read/write current pulses due to the large capacity between the inhibit/disturb drive lines and the read/write drive lines.

Since the memory word length is 36 bits, there are 36 inhibit/disturb generators. It is desirable to be able to manually adjust the current magnitude of all 36 generators by a single control. To accomplish this, despite wide variations in transistor gain, the emitter resistor is used to provide dc stabilization of the circuit. Actually this resistance is made up of the controlled winding resistance of the inductor, L_1. Controlling the current magnitude produced by all 36 drivers is easily accomplished by varying the base bias voltage of transistor Q_1. Fluctuations in current demand from the power supplies due to the pulsing of the inhibit drivers has been eliminated by maintaining a dc current in the generators, thereby simplifying the design of the +16 volt power supply while permitting the +8 volt supply to vary as much as 10 per cent. This is particularly important since the transient current demand would be quite large (6 to 8 amps) making voltage regulation of low voltage-high current supplies extremely difficult.

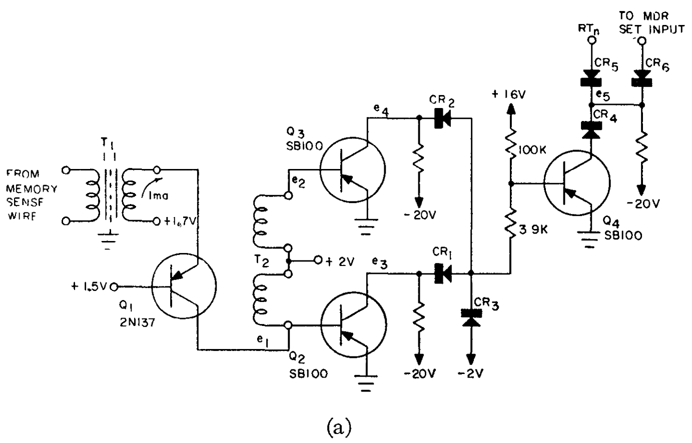
The X and Y read/write current generators produce bipolar half-magnitude current pulses which are coupled to the X and Y coordinate drive busses. The current waveforms have a 1-μsec rise and fall time and a 5-μsec flat portion of approximately 175 ma. The circuitry for the read/write current generator is basically the same as for the inhibit/disturb generator except that two generators and a dual primary transformer are required to produce bipolar current pulses.

One of the special considerations that had to be taken into account in the design of the read/write generator was the response of the read/write transformer. Under certain operating conditions, the load as seen by the transformer may not change for long periods of time. This drive line impedance is relatively low and extends the recovery time of the transformer to a period greater than the interval between current pulses. Because of this, the base reference line of the current pulses would tend to seek different levels in either the plus or minus direction depending upon the interval between memory references and also the degree and direction of unbalance between the read and write pulses. To insure a zero base line at the start of each reference, all of the energy stored in the transformer must be dissipated prior to the next reference. To accomplish this, a secondary line switch was placed in the transformer current return path (Fig. 18, preceding page). The circuit of this switch is the same as the current diverter except that it is normally in the conducting state. At the end of each reference, the switch transistor is cut off for 2 μsec. This presents a fairly high impedance to the transformer secondary winding, and the stored energy dissipates very quickly so that the transformer will be ready for the next reference at the end of each cycle.

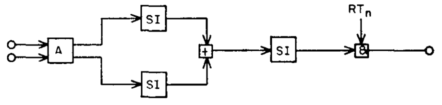
Sense Amplifier

The sense amplifier used in each digit plane control is shown in Fig. 19. A 1:1 balanced transformer, T_1, couples the memory core output signal into the grounded base input stage Q_1. The bias voltages applied cause approximately a 1-ma dc current to flow through the transistor into the auto transformer T_2. The 14 to 18 mv signals are amplified to approximately 0.5 volt based at +0.2 v and clamped by the bases of transistors Q_2 and Q_3. The bipolar pulses are rectified and amplified to the 2 volt level shown. These signals are transmitted to a single inverter to place them in the proper polarity for logical combination with the read strobe signal. The output of transistor Q_4 is a 2-volt signal 2 μsec wide, occurring during the read signal. Waveforms at the various points are shown.

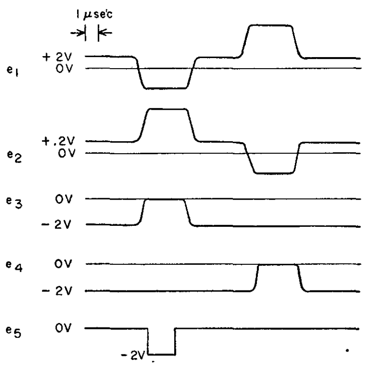
The +0.2 volt reverse bias on transistor Q_2 and Q_3 is sufficient to eliminate all unwanted noise signals which occur at probe time. The wanted signal is sufficient to cause these two transistors to saturate.



(a)



(b)



(c)

Fig. 19—(a) Sense amplifier schematic diagram, (b) sense amplifier logical block diagram, (c) waveforms.

Conclusion

The results achieved in this design clearly indicate that larger and faster all-transistor memory systems are feasible. The power required to operate this memory is approximately 300 w and the space required is approximately 5 cubic feet. This represents a power reduction over an all-tube equivalent of about 10 times and a weight and volume reduction of about 20 times. The performance of the 1200 transistors used in this system has yet to be demonstrated but it is felt that it will be far better than the vacuum-tube equivalent.

Discussion

S. C. Chao (General Electric): What is the accuracy of the driving currents?

Dr. Valenty: Plus or minus 50 milliamperes, or approximately 10 per cent.

E. Slobodzinski (I.B.M.): What is the environment temperature-wise that your system was designed to operate in? What have you considered as the worst case for component and transistor tolerances?

Dr. Valenty: The system is designed to operate up to 90 degrees F; over that it is automatically turned off.

Millimicrosecond Transistor Current Switching Techniques

H. S. YOURKE† AND E. J. SLOBODZINSKI†

Introduction

THERE ARE five major limitations on the speed of transistor switching circuits. These are: 1) carrier storage delays, where the transistors are operated in saturation, 2) the limitations imposed by transistor and circuit capacitances, 3) α cutoff frequency, 4) diffusion or transit-time delay, 5) storage time in associated diodes.

If transistor switching circuits are to have response times limited primarily by the bandwidths of the transistors operating as amplifiers and by diffusion or transit-time delay, it is necessary to avoid operation in saturation. As low collector-to-base voltage generally has a detrimental effect on transistor bandwidth, it is desirable, as well, to avoid operation near saturation.

Where nonsaturating circuits are used, and when transistors having cutoff frequenceis of several hundred megacycles are considered, circuit capacitances become the primary limitation on speed. If we assume a current step into a node, where there is capacitance to ground at the node, the voltage rise time is proportional to the required voltage swing. Therefore, to minimize the effects of circuit and transistor capacitances it is desirable to operate with voltage swings as small as reliability considerations will permit.

A new mode of operation has been developed whereby well-specified currents can be switched reliably with small voltage swings. Transistors are operated well out of saturation, and switching speeds approach that of a grounded-base amplifier driven by a current step. The resulting logical circuits reset their own levels. The circuits place no requirement on the upper limit of α and have a dc stability factor of unity. They have complemented outputs and provide an essentially constant load to all dc power supplies. The circuits are simple and relatively noise free.

The Mode of Operation

The basic principle of the mode of operation presented here is the switching of a constant current into the emitter of a single transistor, or the emitters of a group of parallel transistors, to the exclusion of other transistors whose emitters are tied to the same node. This mode of operation is best illustrated by an examination of the basic building block shown in Fig. 1. If the voltage at the base of the top transistor is permitted to assume a value either slightly more positive or slightly

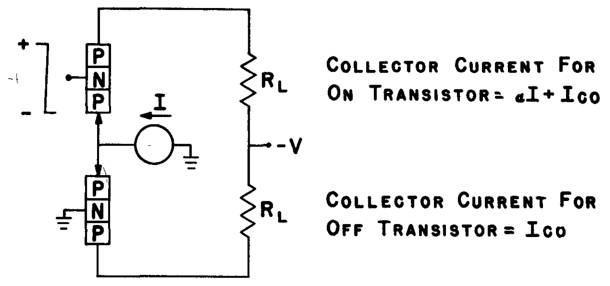

+
-
P
N
P
I
R_L
-V
P
N
P
R_L
Collector Current For On Transistor = $\alpha I + I_{CO}$
Collector Current For Off Transistor = I_{CO}

Fig. 1—Basic transistor block.

more negative then the potential at the base of the bottom transistor (in this case, ground), the current, I, will flow into the emitter of one transistor to the exclusion of the other. A voltage swing of $+0.4$ to -0.4 volts about the reference voltage has been found sufficient to guarantee switching of up to 10 ma in high-frequency experimental drift transistors.

For a positive input voltage whose magnitude is equal to, or greater than, the emitter-to-base voltage drop of the bottom transistor, the bottom transistor will conduct and the top transistor will be off. For a negative input voltage whose magnitude is equal to or greater than the emitter to base voltage drop of the top transistor, the top transistor will conduct and the bottom transistor will be off. The collector current for the nonconducting transistor will be I_{co}, and the collector current for the other will be $I_{co}+\alpha I$.

For a voltage step applied at the input, the transient behavior of the circuit approaches that of a grounded-base amplifier driven from a current step. There is simultaneous switching of emitter currents in the two transistors. The outputs of the transistors are complementary. If a complemented output is not desired, the bottom transistor may be replaced by a diode. As noted above, the dc stability factor is unity.

Coupling Techniques

There are several techniques for coupling basic building blocks or the resulting logical circuits. One technique uses alternate *p-n-p* and *n-p-n* blocks, directly coupled. Another uses low impedance voltage translating blocks as coupling devices.

Two basic building blocks, one a *p-n-p* block and the other an *n-p-n* block, are shown in Fig. 2. Both blocks are 6-ma current switches. An unloaded input voltage swing of ± 0.6 volt with respect to the reference voltage at the base of the bottom transistor is required for reliable switching in either block. The voltage levels at the out-

† Research Center, IBM, Poughkeepsie, N. Y.

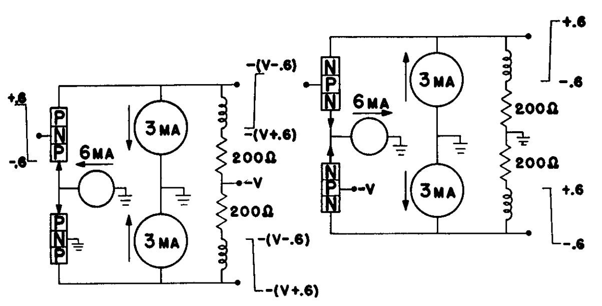


Fig. 2—*N-P-N* and *p-n-p* basic transistor blocks with input and output levels shown to illustrate compatibility.

puts are ideal for driving blocks of the opposite kind.

The 3 ma current sources in the collector circuits are not essential. Their use, however, permits the collector load resistors to be returned to the reference voltage of the succeeding stages, thereby reducing the effects of noise on the reference voltage lines. The peaking coils provide a degree of transient overdrive and improve the speed and cascading factor. Since collector voltage swings are small, the transistor may operate at a fairly constant collector voltage whose magnitude is chosen to provide maximum transistor bandwidth. Generally, the collector voltage would be as large as breakdown voltage and power dissipation considerations would permit.

In Fig. 3, a *p-n-p* block is shown driving a number of *n-p-n* blocks located at a relatively large distance. The peaking coil and collector load resistor have been moved from the *p-n-p* block to the vicinity of the *n-p-n* blocks being driven. Since current is being transmitted, rather than voltage, series parasitic elements along the line, such as contact-resistance and inductance, have no effect on the dc levels at the end of the line and have little effect on the transient behavior. The voltage at the inputs to the *n-p-n* blocks will be either 0.6 volt more positive or 0.6 volt more negative than the reference voltage; and the system is virtually insensitive to noise on the reference voltage. Since impedance levels at all nodes are 200 ohms or less, the system is relatively free of noise due to capacitive coupling.

In the example shown, the long line is terminated in a dc impedance of 200 ohms. If larger currents were to be switched the line could be terminated in a proportionally lower impedance, since the required voltage swing would be essentially the same. This technique of placing the load resistor at the end of the long line should lend itself to coupling through low characteristic impedance transmission lines.

The use of voltage translating blocks for coupling basic building blocks or the resulting logical circuits is illustrated in Fig. 4. The black box contains a battery, or any device that simulates a battery, and provides an essentially constant voltage drop equal to the desired nominal collector voltage. With the upper transistor cutoff, I ma flows through the black box. The source on the load side of the box provides a current of I–3 ma,

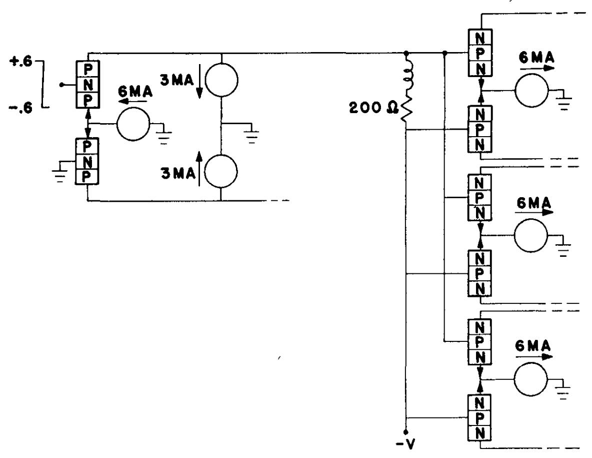


Fig. 3—A *p-n-p* block driving a number of remote *n-p-n* blocks.

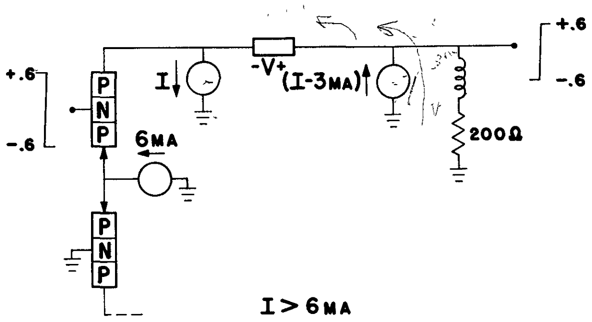


Fig. 4—Application of a voltage translating block as a means of coupling.

and 3 ma flows out of the load resistor, establishing an output potential of -0.6 volt.

When the upper transistor is conducting 6 ma, I–6 ma flows through the box, and since I–3 ma is provided on the load side of the box, 3 ma flows into the load resistor, establishing an output potential of $+0.6$ volt. The output of this block may drive either *p-n-p* or *n-p-n* blocks.

Although this coupling technique requires more components and consumes more power than does the coupling of alternate *p-n-p* and *n-p-n* blocks, the advantages of low impedance levels and relative freedom from noise are retained.

Typical Logical Circuits

And-Or Circuits

Application of the current switching principle described here results in a variety of circuits capable of performing many logical functions. A description of a few such circuits will illustrate the versatility of the system. A typical building block is shown in Fig. 5. Using the notation that the output of a conducting transistor and the input enabling it to conduct are logical ONE's, this circuit may be called an *N*-way complemented OR circuit. This notation is convenient in dealing with the coupling of alternate *p-n-p* and *n-p-n* logical blocks.

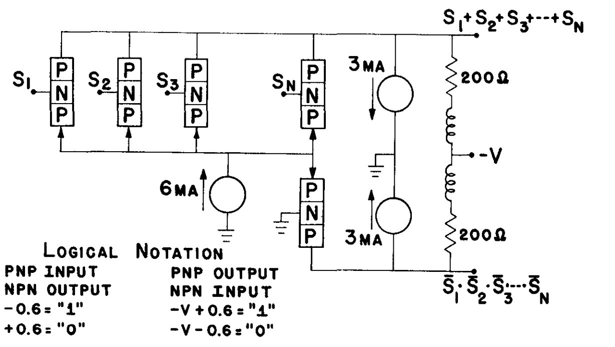


Fig. 5—*N*-way complemented OR circuit.

Where any or all of the input signals to this circuit are logical ONE's, the current through the parallel combination of the top transistors is 6 ma, and the current through the bottom transistor is zero. The bottom transistor conducts 6 ma only when all inputs are logical ZERO's. For this case, the combined current through the top transistors is zero. Since all signals have their complements available, the circuit can perform AND or OR operations on *N* signals.

Fig. 6 is a photograph of an input waveform and the output waveforms of a 3-way complemented OR circuit, with two inputs held at logical ZERO. The input waveform, a ten megacycle square wave, is shown at the top. The two output waveforms are shown below. The noninverted waveform is the AND output and the inverted waveform is the OR output. The circuit was operating as a 6-ma current switch, using drift transistors. The oscilloscope was synchronized with the input signal.

Fig. 7 shows the response of the same circuit when two complementary inputs are applied while the third is a logical ZERO. Ideally, the two outputs should remain constant. The AND output, shown in the center, is a logical ZERO. The OR output, shown at the bottom, is a logical ONE. Some logical noise does appear at the OR output when the 6-ma current is switching between two of the three parallel transistors.

Fig. 8 compares the outputs of a 3-way complemented OR circuit, shown at the top, and a 10-way complemented OR circuit, shown below, when driven from the same signal. Only slight degeneration in both waveshape and delay is observed, indicating that the useful number *N* for the *N*-way complemented OR circuit is quite large.

To obtain an indication of the cumulative delay through logical stages, a circuit was constructed having four 3-way complemented OR circuits in sequence, each with a cascading factor of three. This circuit is shown in Fig. 9. The blocks were 6-ma current switches using drift transistors, and could be connected through the inverted or the noninverted outputs. The delay through the system is measured between any output on the right and the input to the first three blocks.

Fig. 10 is a photograph of the input and output waveforms of the system. At the top are the input and output waveforms when the blocks are connected through

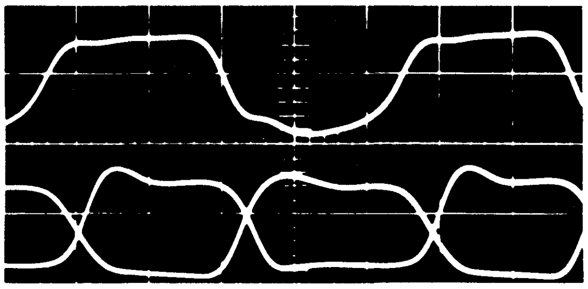


Fig. 6—Three-way complemented OR circuit. Two inputs are logical ZERO's, third input is the top waveform. Lower waveforms are outputs. (Hor.: 20 musec/cm, vert.: 1 volt/cm.)

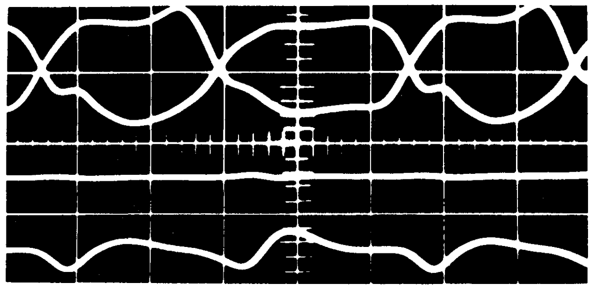


Fig. 7—Three-way complemented OR circuit. One input is a logical ZERO, top waveforms are two complementary inputs. Center waveform is AND output. Lower waveform is OR output. (Hor.: 20 musec/cm, vert.: 1 volt/cm.)

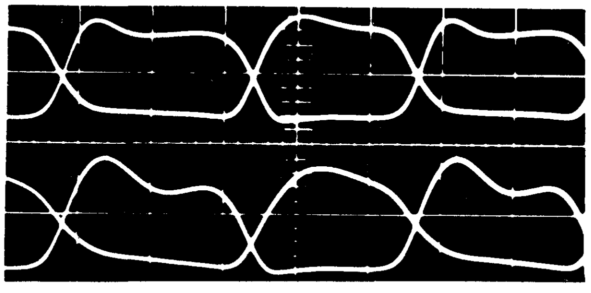


Fig. 8—Outputs of a ten-way complemented OR circuit, lower waveforms, compared with the outputs of a three-way complemented OR circuit, upper waveforms, driven by the same input signal. (Hor.: 20 musec/cm, vert.: 1 volt/cm.)

the inverted outputs. Four inversions result in an inphase output. The lower waveforms are the input and output waveforms when the blocks are connected through the noninverted outputs. The total delay through the four blocks in both cases is between 35 and 40 millimicroseconds.

Exclusive OR

Another important logical block is the exclusive OR circuit. A six-transistor complemented exclusive OR circuit is shown in Fig. 11. This circuit makes use of the fact that all signals have their complements available.

The combined outputs of the bottom transistors will be 6 ma for the exclusive OR statement and will be zero for the two other possible statements. The combined outputs of the top transistors will be 6 ma for the exclusive OR statement and 12 ma for the two other possible statements.

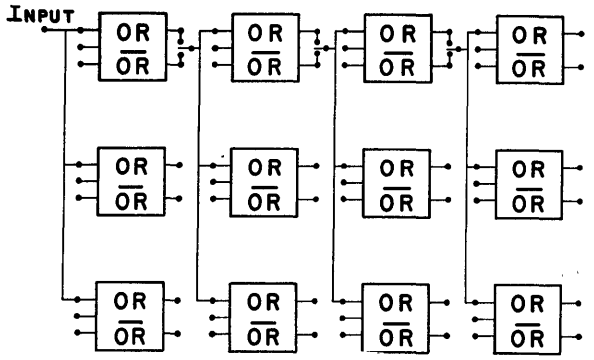


Fig. 9—Block diagram of four complemented OR circuits in sequence, with a cascading factor of three.

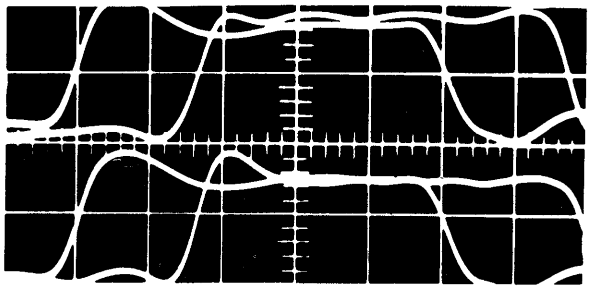


Fig. 10—Input and output waveforms of four sequential three-way complemented OR circuits with a cascading factor of three. Top waveforms are input and output of the system with blocks connected through inverted outputs. Lower waveforms are with the blocks connected through noninverted outputs. (Hor.: 20 musec/cm, vert.: 1 volt/cm.)

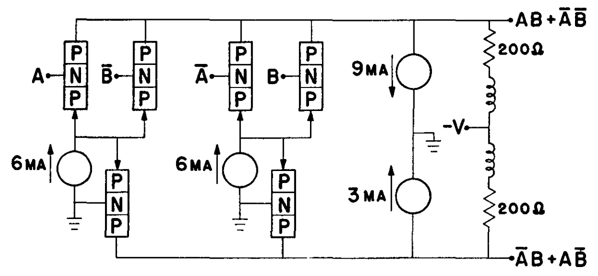


Fig. 11—Six-transistor, complemented, exclusive OR circuit.

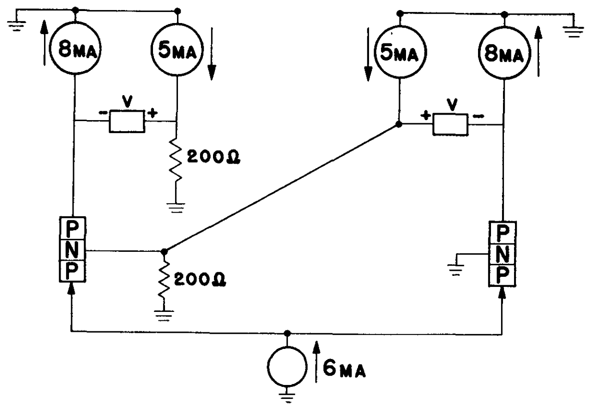


Fig. 12—Simple two-transistor trigger or flip-flop.

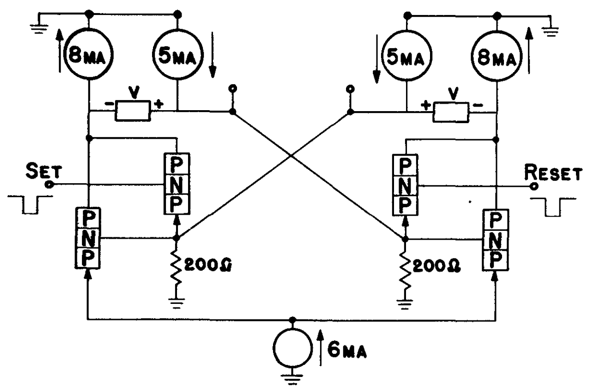


Fig. 13—A typical trigger circuit for use in set and reset operation using *p-n-p* emitter followers for pullover.

The 3-ma and 9-ma current sources are arranged so that the currents into the 200-ohm terminating resistors switch between plus and minus 3 ma. The switching speeds and logical noise for this circuit are essentially the same as for the complemented OR circuit.

Triggers

This mode of operation, using small voltage swings to switch well defined currents, lends itself readily to the design of triggers or flip-flop circuits, the simplest of which is shown in Fig. 12. The circuit is bistable by virtue of the common-emitter current source and the feedback from one collector to the opposite base. The circuit may be triggered by pulsing the base of the transistor which is tied to the opposite collector. The lack of symmetrical inputs makes it difficult to employ conventional pullover techniques without changes in the collector to base feedback loop and the base circuit of the normally grounded base transistor.

Fig. 13 illustrates a typical trigger circuit employing feedback to both bases with *p-n-p* emitter follower pullovers. The 200-ohm resistors are the loads normally found in the collectors of a 6-ma transistor block as described previously. The base of the ON transistor will be biased at -0.6 volt while the base of the OFF transistor will be biased at $+0.6$ volt. The pullover transistor bases are normally at $+0.6$ volt. During a set or reset operation, the appropriate pullover base is brought to -0.6 volt. This causes the emitter to follow the input with some level shift. However, the emitter load of the pullover is brought sufficiently past the point where regeneration begins so that the set or reset operation is completed with a minimum of delay.

A degree of overdrive is obtained by emitter follower action, and also because the collector of the active pullover transistor is tied to the collector of the transistor whose base is being switched. Hence, α times the emitter current of the pullover transistor appears at the collector of the transistor being turned on, thereby enhancing the output signal at that point.

Fig. 14 shows an extension of the technique to the use of *n-p-n* emitter followers as pullover transistors. Here the pullover bases are normally biased at -0.6 volt

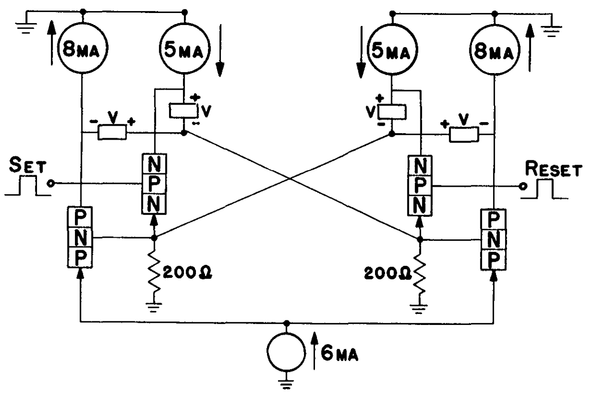


Fig. 14—A typical trigger circuit for use in set and reset operation using *n-p-n* emitter followers for pullover.

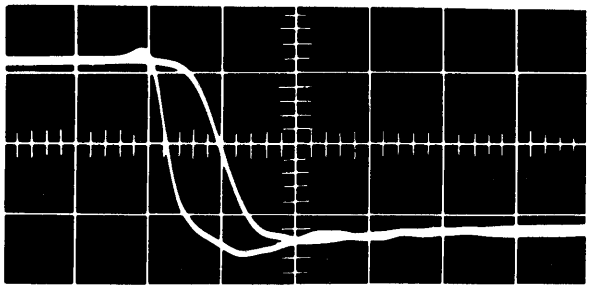


Fig. 15—Input and output waveforms of three sequential triggers using *p-n-p* emitter followers as pullover transistors. The noninverted output signals were used for coupling. (Hor.: 20 musec/cm, vert.: 0.5 volt/cm.)

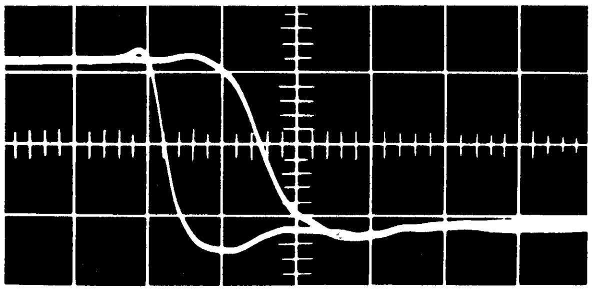


Fig. 16—Input and output waveforms of three sequential triggers using alternate *p-n-p* and *n-p-n* emitter followers as pull-over transistors. The inverted output signals were used for coupling. (Hor.: 20 musec/cm, vert.: 0.5 volt/cm.)

and the pullover collectors are tied to the collectors of the transistors being switched. Proper collector bias is obtained for the *n-p-n* transistors by the use of a translating block in the positive current source. The operation of this circuit is analogous to the circuit of Fig. 13 except that positive swings change the state of the trigger.

The delays and rise times of the emitter following pullover triggers are of the same order of magnitude as those for the other logical blocks described. Fig. 15 is a photograph of the input and output waveforms of three sequential triggers using *p-n-p* emitter followers as pullover transistors. The noninverted trigger outputs were used for coupling, giving a total delay of approximately 15 millimicroseconds. This delay is essentially that of the emitter followers and is caused, in part, by the slope of the input waveform.

The inverted output of the trigger will have delays somewhat greater than that of the noninverted output, since in this case the delay is dependent on the pullover transistor and the trigger transistors, as well as on the slope of the input waveform. These delays are of the same order of magnitude as the delay through a simple transistor block. Fig. 16 shows the delay through three sequential triggers, employing alternate *p-n-p* and *n-p-n* emitter followers as pullover transistors, in which the inverted outputs were used for coupling. From Fig. 16, the total delay is approximately 24 millimicroseconds with a delay per trigger of approximately 8 millimicroseconds.

ACKNOWLEDGMENT

The authors wish to acknowledge the contributions of other members of the high-speed transistor circuit group, and members of the semiconductor device group. Both groups are at the IBM Research Center, Poughkeepsie, New York.

Discussion

J. R. Braun (Electro Data): What is used for the current sources shown?

S. Sloan (Norden Ketay): Will you please describe the current sources used?

David Zeheb (General Electric): What is the internal impedance of these sources?

Mr. Yourke: Transistors can be used as current sources. For the circuit shown, the resistance value is of the order of 500 ohms.

John Teska (North American): What is a "voltage translating block"?

Mr. Yourke: The voltage translating block is a device which may be biased as to provide an essential, constant voltage drop for a relatively large current swing. This device is a typical type; however, for these circuits a very well-specified voltage swing is not essential, but is conceivable on others that are not so specified.

J. Foulkes (Bell Telephone Labs.): Would you give us a few details of the transistors used?

Mr. Yourke: The transistors used were of IBM manufacture. Their cut-off frequency speck is above 70 megacycles; some go up as high as 300 megacycles. However, for the circuits shown, the speeds are limited primarily by incidental specifications, and very little difference in time is observed where transistors have a cut-off frequency of more than 300 megacycles.

J. P. Brosius, Jr. (Autonetics): Is IBM making any specific use of the exclusive "or" circuit, and, if so, what uses?

Mr. Yourke: The exclusive "or" circuit is used exclusively at IBM. A great deal of use is made of the "or" circuit in adding operations.

R. C. Spriestersbach (Librascope): Have you worked with silicon transistors in this type of circuit? If so, how fast?

Mr. Yourke: No, we have not.

David Zeheb (General Electric): Do you safeguard against an open circuit, and if not, what value of voltage will result in case of an open circuit?

Mr. Yourke: The only place where any problem could be serious is in the voltage translating block with *n-p-n* and *p-n-p* transistors. There is very little danger of excessive voltage from open circuit at any point.

The Utilization of Domain-Wall Viscosity in Data-Handling Devices

VERNON L. NEWHOUSE†

The Magnetic Inertia Effects

Experimental Account

During the investigation of the high-speed switching of $\frac{1}{8}$-mil grain oriented 4-79 Molybdenum Permalloy tape a group of effects were discovered which do not seem to have been previously described in the literature. These effects will be referred to as the *magnetic inertia effects*. They appear to be associated with the viscosity of the magnetic domain boundaries and can be demonstrated in ferrites as well as in Molybdenum Permalloy. This paper will be concerned with the description and utilization of the effects in 4-79 Molybdenum Permalloy as it is this material which has been chiefly investigated to date. The Magnetic inertia effects can be described under the three following headings.

Nondestructive Read-Out: This effect has been demonstrated in $\frac{1}{8}$-mil as well as $\frac{1}{4}$-mil grain oriented tape made of 4-79 Molybdenum Permalloy. It was found that the application of magnetizing pulses much larger than the coercive force did not give rise to permanent changes of magnetization provided that the duration of the pulses was sufficiently short. This effect is demonstrated in Fig. 1 (next page) which shows the waveforms resulting from the application of a continuous train of 0.1 microsecond half sine-wave current pulses through suitable magnetizing windings to a toroid consisting of 5 wraps of $\frac{1}{8}$-mil material. The coercivity of this material was approximately 0.08 oersted. The peak height of the magnetizing pulses shown in Fig. 1 is 0.63 oersted. This is close to the maximum amplitude of 0.1-microsecond half sine-wave pulses which can be applied to this material without significantly affecting the permanent state of magnetization.

Size Anomaly of Nondestructive Read-Out Signal: The minimum amplitude of 0.1-microsecond pulses required to switch completely the core under discussion was 2.8 oersteds. Waveforms accompanying complete switching are shown in Fig. 2. Comparison with Fig. 1 shows that the reversible magnetization changes occurring during nondestructive read-out consist of a relatively large fraction of the total magnetization change obtainable on completely switching the core. The nondestructive read-out signals are approximately symmetrical about the base line in $\frac{1}{8}$-mil material and their amplitude is strongly dependent on the state of remanence. In a representative case such as that shown in Fig. 1, the ratio between the peak amplitudes of the "one" and "zero" output signal is of the order of 3:1, and the peak of the voltage pulse associated with the nondestructive read-out in the "one" state is as high as 17 per cent of the voltage pulse associated with complete flux reversal.

A High Field Threshold: In the type of core described above having a coercivity of approximately 0.08 oersted, the maximum amplitude of 0.1-microsecond pulses which do not cause a permanent change of flux is of the order of 0.63 oersted. This is less than $\frac{1}{4}$ of the pulse height required for complete switching. To use the inertia effects for writing into a random-access memory it would be useful to find some way of applying pulses of at least $\frac{1}{2}$ the height required for complete switching without destroying the information content of the core. A means of doing this exists. It is found that pulses of more than half the amplitude required for complete switching can be applied without causing a permanent change of state, provided that each pulse is followed by an opposite polarity pulse of similar amplitude. Alternatively a *pair* of positive pulses can be followed by a *pair* of negative pulses. The time interval between the pulses can be of arbitrary length. The permissible difference in amplitude is of the order of 7 per cent at 1.8 oersteds, and becomes larger as the pulses become smaller. Output signals resulting from nondestructive read-out of this type are shown in Fig. 3.

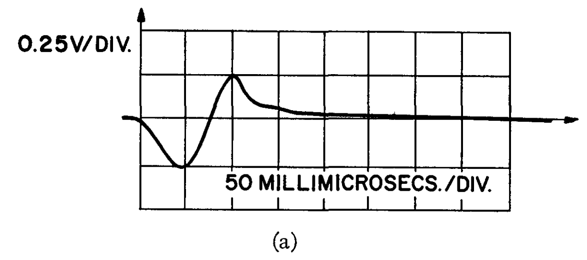
The technique of current amplitude coincidence for core selection in conventional memories employs the core coercivity as a threshold mechanism. The switching speed of cores selected in this fashion is limited because the total switching field applied cannot be made greater than twice the coercivity. The results described above indicate the existence of a threshold field much higher than the coercivity.

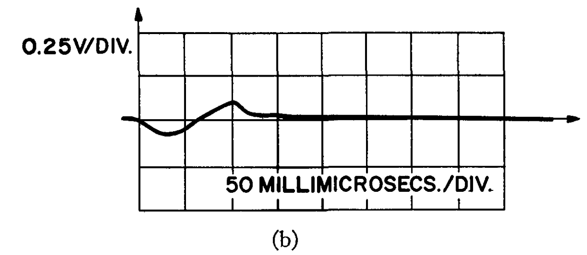
This makes it possible to use amplitude coincidence for core selection using much larger fields than the coercivity and consequently attaining much higher switching speeds.

Some numerical results which have been abstracted from Figs. 1 to 3 are summarized in Table I, p. 75. It is interesting to compare the flux change associated with complete switching—19-volt millimicroseconds/turn with the complete reversible 2.4 volt millimicroseconds/turn which occurs during nondestructive read-out with pulses in one direction.

The flux change which occurs during the application of a sensing pulse of 1.8 oersteds is shown to be 9.1-volt millimicroseconds/turn. Integration of the waveforms of Fig. 3 shows that approximately 43 per cent of this flux change is elastic, *i.e.*, reversible, and exists only during

† Radio Corp. of America, Camden, N. J.





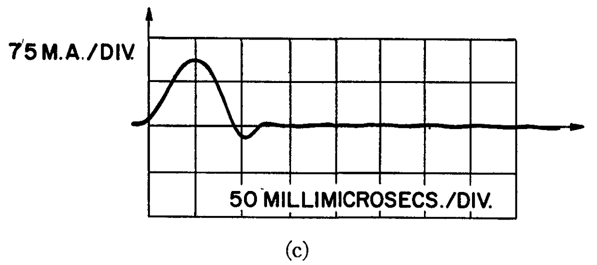
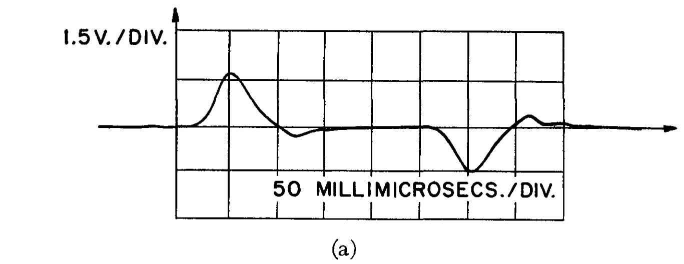
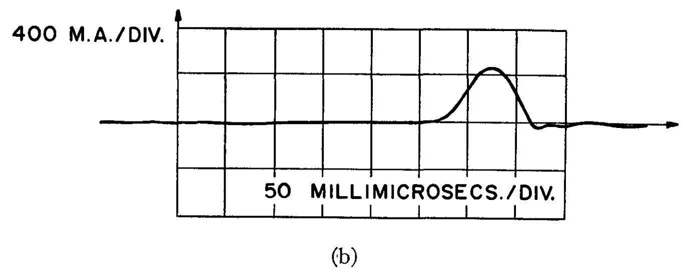


Fig. 1—Nondestructive read-out in ⅛-mil 4-79 Molybdenum Permalloy.
Core construction: 5 wraps of ⅛-mil tape, $\frac{1}{32}$ inch wide, ceramic bobbin, ⅛ inch od.
(a) Voltage across 5-turn sense winding: core at negative remanence.
(b) Voltage across 5-turn sense winding: core at positive remanence.
(c) Negative magnetizing pulse through 5-turn drive winding. Rep. rate: 2 kc.





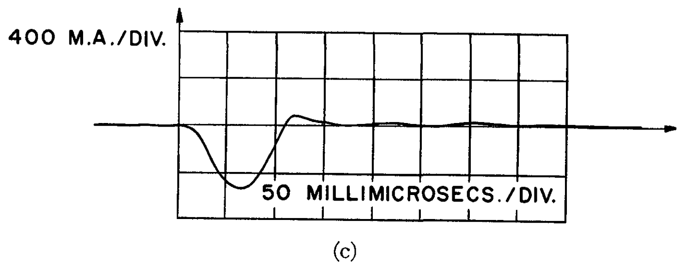
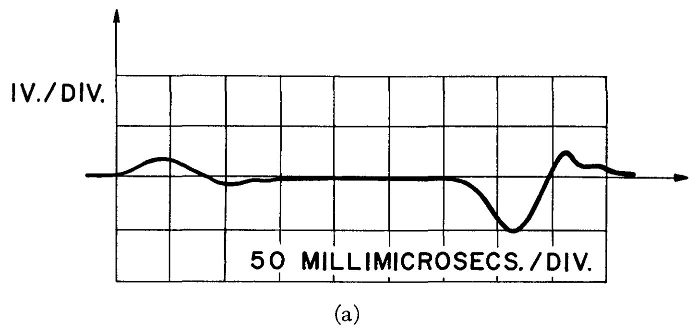


Fig. 2—Complete switching of ⅛-mil 4-79 Molybdenum Permalloy.
Core construction: As in Fig. 1.
(a) Voltage signal across 5-turn sense winding.
(b) Set pulse through 5-turn winding. (Minimum amplitude required for complete switching.)
(c) Reset pulse through 5-turn winding.

the application of the sensing pulse. The other 57 per cent is inelastic and remains after the termination of the sense pulse. This inelastic flux change can be cancelled out by a pulse which is within 7 per cent of the initial pulse in peak amplitude but in the opposite direction. The fact that the net flux excursion over a whole cycle is zero is proved by the fact that the nondestructive read-out effect is maintained even when positive and negative pulses are applied indefinitely.



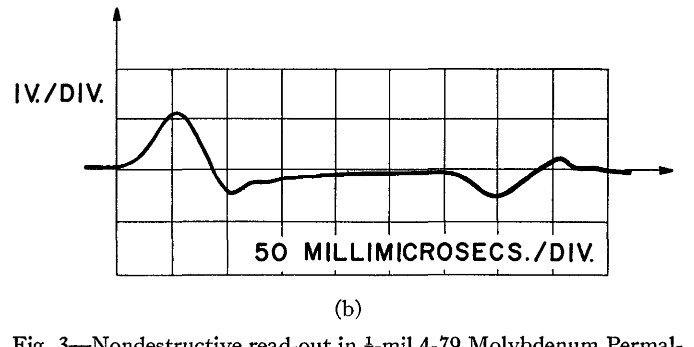


Fig. 3—Nondestructive read-out in ⅛-mil 4-79 Molybdenum Permalloy using symmetrical 0.1-microsecond half sine wave pulses of 1.8-oersteds peak amplitude.
Core construction: as in Fig. 1.
(a) Voltage signal across 5-turn sense winding. Core at positive remanence.
(b) Voltage signal across 5-turn sense winding. Core at negative remanence.

Physical Mechanisms

The magnetic inertia effects can be explained at least qualitatively in terms of existing domain theory. Following a discussion of each of the three inertia effects, the ideas presented will be examined in the light of recent results obtained by other workers from the study of thin films. It will be seen that several of the mechanisms used to account for the magnetic inertia effects in grain oriented tape have been shown to be of importance in the switching behavior of thin films.

Nondestructive Read-Out: The fact that it is possible to apply an indefinitely large number of very short mag-

TABLE I

REVERSIBLE AND IRREVERSIBLE SWITCHING IN $\frac{1}{8}$ MIL GRAIN ORIENTED 4-79 MOLYBDENUM PERMALLOY USING 0.1-MICROSECOND HALF SINE-WAVE PULSES

Mode of Operation	Driving Pulse	"1" Output Voltage Pulse	"0" Output Voltage Pulse	Peak Flux Change
Destructive Switching	2.9 oe*	0.30 v/turn	0.07 v/turn	19-millimicrosec volts/turn
Nondestructive Read-Out with Pulses of One Polarity	0.63 oe†	0.050 v/turn	0.016 v/turn	2.4-Millimicrosec volts/turn†
Nondestructive Read-Out with Symmetrical Pulse Excitation	1.8 oe	0.20 v/turn	0.07 v/turn	9.1-Millimicrosec volts/turn‡

* Minimum excitation for complete switching.
† Peak excitation which can be used in this mode.
‡ This represents the flux change which occurs during the application of a pulse tending to reverse the magnetization.

netizing pulses considerably exceeding the coercive force of $\frac{1}{8}$-mil or $\frac{1}{4}$-mil grain oriented Molybdenum Permalloy tape without causing a cumulative change of the state of magnetization can be explained in terms of the domain-wall viscosity and surface tension.

It seems likely that the existing domain walls are moved over such a relatively small distance during the application of the nondestructive read-out pulse that they fall back to their original position after the cessation of the pulse. The assumption that the movement of domain walls over small enough distances is not accompanied by irreversible magnetization changes is supported by the well known fact that the application of sufficiently weak fields of arbitrary duration to conventional magnetic materials will not result in irreversible magnetization changes. This is true even in portions of the hysteresis loop where most of the magnetization change takes place by domain-wall motion.

It can be calculated that the distance moved by a domain wall under the influence of the maximum 0.1-microsecond pulse of 0.63 oersted which gives nondestructive read-out in the case of $\frac{1}{8}$-mil permalloy is approximately five times its own thickness. This result indicates that the mechanism assumed for the nondestructive read-out effect is plausible, since studies of the Barkhausen effect in related materials indicate that irreversible changes only become important as a domain wall moves through a distance which is between one and ten times its own thickness.[1]

The nondestructive read-out signal from $\frac{1}{8}$-mil material shown in Fig. 1(a) shows a slight exponential "tail." This feature is accentuated in $\frac{1}{4}$-mil material and is probably associated with eddy currents. The calculated decay time constant of the eddy current field agrees to within an order of magnitude with the experimentally observed values. The value of permeability used in this calculation is 4000 cgs units, a value which corresponds to the flux changes observed during nondestructive read-out.

It has been pointed out that nondestructive read-out is possible with pulses as large as half the amplitude required to switch the core completely, provided that each positive pulse or pair of pulses is followed at an arbitrarily long time interval by an equal number of negative ones. In this case the application of the first pulse tending to switch the material results in a degree of irreversible wall movement. It is clear that the reabsorption pulse moves the domain walls to a position sufficiently close to their remanence location for no cumulative changes to occur under the influence of repeated pairs of pulses.

Size Anomaly of Nondestructive Read-Out Signal: Having shown that nondestructive read-out using short pulses is possible because the domain walls move over a relatively short distance, it is necessary to explain why the reversible changes accompanying nondestructive read-out with pulses in one direction are as large as 12 per cent of the major loop magnetization change. Moreover, the reversible component of the flux change associated with nondestructive read-out using symmetrical excitation is as high as 19 per cent of the flux change associated with complete switching. These results cannot be accounted for on the basis of domain wall movement alone.

The major contribution to the reversible magnetization changes during nondestructive read-out by short pulses is believed to be that due to spin rotation; *i.e.*, the coherent rotation of the magnetization of a whole domain over a small angle.

This is a process which can be represented by a very simple mathematical model which has been studied rather exhaustively because it is of considerable importance in the operation of small particle permanent magnets.[2] Magnetization by rotation is a process which is known to be reversible provided the rotation is over an angle less than a critical value. It is difficult to calculate the value of applied fields at which spin rotation becomes significant, because the magnetic anisotropy constant for Molybdenum Permalloy is rather uncertain. It can however be measured experimentally by applying a field at right angles to the easy direction of magnetization and turns out to be less than 2 oersteds. This is in agreement with the fields of the order of

[1] R. S. Tebble, I. C. Skidmore, and W. D. Corner, "The Barkhausen effect," *Proc. Phys. Soc. A*, vol. 63, pp. 739–761; 1950.

[2] E. C. Stoner and E. P. Wohlfarth, "A mechanism of magnetic hysteresis in heterogeneous alloys," *Phil. Trans.*, vol. A 240, pp. 599–644; May, 1948.

1 oersted which are found to give nondestructive read-out. Further confirmatory evidence is provided by the work of Conger who has shown that magnetization reversal takes place by rotation rather than wall movement in thin films of 80-20 nickel-iron for applied fields larger than a few oersteds.[3]

A further contribution to the reversible magnetization changes occurring during the nondestructive read-out pulse may be due to the creation of reversal domains around imperfections in the material such as grain boundaries. These would be reabsorbed after the termination of the pulse.

The High Field Threshold: It has been stated that the magnetization changes due to 0.1-microsecond pulses have been shown to be mainly reversible below 1.8 oersteds and completely irreversible above 2.8 oersteds. The existence of this second threshold for core switching is a further indication that spin rotation rather than wall movement is taking place since magnetization by wall movement has a threshold at fields of the order of the coercivity which in this case is about 0.1 oersted.

The fact that spontaneous rotation throughout the material occurs when the second threshold field between 1.8 and 2.8 oersteds is exceeded indicates that spin rotation may occur at the lower fields used for nondestructive read-out, in areas surrounding imperfections in the material such as grain boundaries. This type of spin rotation around imperfections could lead to the formation of the transient reversal domains mentioned in connection with the nondestructive read-out effect.

The results of the above section can be summarized as follows.

The phenomenon of nondestructive read-out associated with the application of very short intense magnetizing pulses is attributed to the fact that existing domain walls are moved over distances within their "elastic limit." Any extra domains created during a nondestructive read-out pulse are reabsorbed after its termination.

The reversible magnetization change occurring during nondestructive read-out is ascribed to the reversible movement of existing domain walls and to temporary coherent rotation of the direction of magnetization in areas large compared to the wall thickness. The creation of temporary reversal domains around imperfections in the material may also make a contribution.

The second threshold effect occurs at fields which have been shown to cause magnetization reversal by rotation in films and is therefore identified as corresponding to the onset of irreversible rotation effects.

Review of Related Work

Many nondestructive read-out techniques have been described which involve the use of cores with special geometries.[4–8] A nondestructive read-out technique which uses cores having a conventional geometry has been described by Widrow.[9] This makes use of the sense of the curvature of the hysteresis loop near remanence. The output obtained is relatively small in amplitude.

Haynes[10] showed several years ago that the application of microsecond pulses larger than the coercivity to metal tape cores having a switching time of the order of milliseconds produces a reversible flux change whose *duration* is dependent on the state of remanence. The switching of the cores investigated was subject to heavy eddy current damping and the nondestructive read-out phenomena observed were interpreted in this light.

Eddy-current damping does not play an important part in the materials reported on in the present paper. This makes it possible to demonstrate and utilize the dynamic properties of the domain walls and spins to a greater extent than would otherwise be possible.

Applications

Some of the possible data-handling applications of the two types of nondestructive read-out, and of the second threshold effect will now be briefly described.

Magnetic Indicator

The order of magnitude of the nondestructive read-out obtainable with the high excitation possible when symmetrical drive is used is demonstrated by the device shown in Fig. 4. In this deliberately simple circuit the nondestructive read-out signals from a conventional tape core are rectified and used to operate a forward biased neon bulb. The status of the neon indicator shows whether the core being sensed is in a state of positive or negative remanence. In a practical case the discrimination of the output signal could be increased by the use of a bucking core. Alternatively the diode could be replaced with a transistor used as a combined rectifier and amplifier.

Magnetic Switch

Nondestructive read-out has been applied to a channel-selecting magnetic switch which embodies the current-steering technique proposed by Karnaugh.[11] The

[3] R. L. Conger, "High frequency effects in magnetic films," paper presented at the AIEE Conf. on Magnetism and Magnetic Materials, Boston, Mass.; October, 1956.

[4] J. A. Rajchman and A. W. Lo, "The transfluxor—a magnetic gate with stored variable setting," *RCA Rev.*, vol. 16, pp. 303–311; June, 1955.

[5] R. L. Snyder, "Magnistor circuits," *Electronic Design*, vol. 3, pp. 24–27; August, 1955.

[6] D. A. Buck and W. I. Frank, "Nondestructive sensing of magnetic cores," *Comm. and Electronics*, no. 10, pp. 822–830; January, 1954.

[7] R. Thorensen and W. R. Arsenault, "A new nondestructive read for magnetic cores," *Proc. West. Joint Comp. Conf.*, pp. 111–116; March, 1955.

[8] A. Papoulis, "The nondestructive read-out of magnetic cores," Proc. IRE, vol. 42, pp. 1283–1288; August, 1954.

[9] B. Widrow, "A radio frequency nondestructive read-out for magnetic core memories," IRE Trans., vol. EC-3, pp. 12–15; December, 1954.

[10] M. K. Haynes, "Magnetic cores as elements of digital computing systems," Ph.D. thesis, Univ. of Ill.; August, 1950.

[11] M. Karnaugh, "Pulse switching circuits using magnetic cores," Proc. IRE, vol. 43, pp. 570–584; May, 1955.

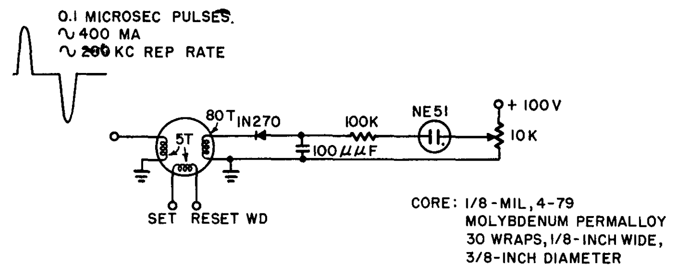


Fig. 4—Nondestructive read-out used to operate neon indicator.

principle of the circuit is illustrated in Fig. 5. Each core is provided with set and reset windings which ensure that only one core is set at a time. These are not shown in the figure.

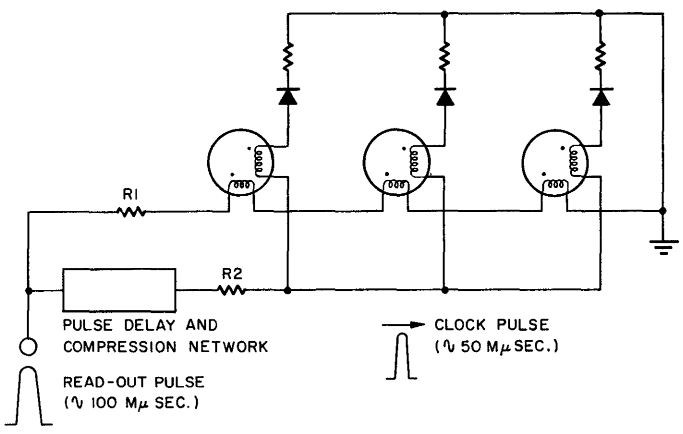

R1
R2
PULSE DELAY AND COMPRESSION NETWORK
CLOCK PULSE (∿ 50 Mμ SEC.)
READ-OUT PULSE (∿ 100 Mμ SEC.)

Fig. 5—Application of nondestructive read-out to current steering switch.

Nondestructive read-out current pulses are applied through R1 in such a direction as to tend to reset the cores. The output windings of the unselected cores produce voltage pulses similar to those shown in Fig. 1(b). The selected core produces a voltage output similar to that shown in Fig. 1(a). The initial output pulse from this core serves to bias off the diodes associated with the unselected cores and "steers" the clock pulse through its own output winding into the selected load. In the circuit shown the clock pulse is produced by delaying and compressing the read-out pulse. Following the termination of the drive pulse the selected core recovers to its "set" state in a fraction of a microsecond. A further read-out pulse can then be applied.

The arrangement of the diodes ensures that no current can flow through any of the loads in the absence of a clock pulse, *i.e.*, during the set and reset of the cores or during their recovery from read-out.

In this circuit the provision of bucking cores in series with each switch core is not required since the operation of the switch ensures that all the sensing current is diverted through the output winding exhibiting the largest voltage pulse. Delay-line effects must however be carefully controlled in the design.

Shift Register with Permanent Output Indication

A one-core-per-bit shift register[12] which is well suited to nondestructive read-out operation is shown in Fig. 6. Point P is normally held at a positive potential. The application of a current pulse through the advance current winding resets all the cores which were previously set, and in doing so, charges the associated intermediate storage capacitors positive. Immediately after the termination of the advance pulse, point P is pulsed negative, thus discharging any of the intermediate storage capacitors which were charged positive in the previous part of the cycle. In discharging a particular capacitor, the subsequent core is set to a state of positive remanence. In Fig. 6 the shift register has been modified to provide a permanent indication of its contents. A sensing-current source has been attached to the advance winding as shown and each intermediate storage capacitor is connected to a neon indicator through a large resistor R. The other end of the neon indicators is taken to a negative potential so as to minimize the inverse voltage required across the diodes of the circuit.

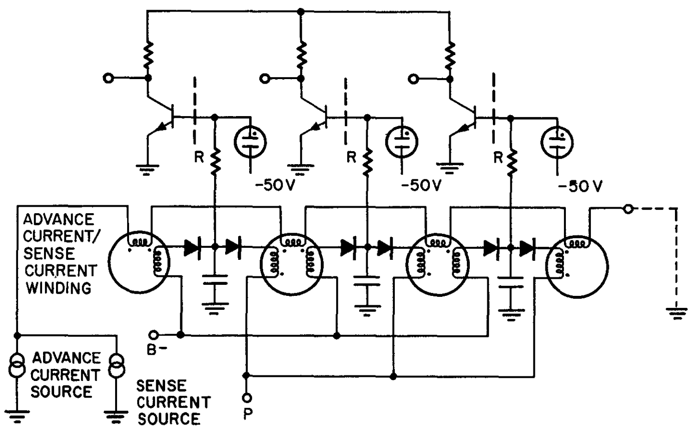


Fig. 6—One-core-per-bit shift register using nondestructive read-out to provide permanent output.

Between the application of advance pulses the sense pulses which can remain "on" continuously, charge the capacitors which follow cores which are in a "set" state to a positive potential and excite the associated neon. A continuous indication of the contents of the shift register is thereby provided. If the neon indicators are replaced by transistor or vacuum-tube amplifiers as indicated in the diagram, other computing or indicating devices can be driven.

It is noteworthy that the shift register conversion proposed, to provide a continuous indication of its content, is achieved with a minimum of added components—one resistor and neon indicator per stage, one current source of pulses and an extra bias supply for the neon indicators. In particular, no extra windings are required.

[12] V. L. Newhouse and N. S. Prywes, "High-speed shift registers using one core per bit," IRE TRANS., vol. EC-5, pp. 114–120; September, 1956.

Random-Access Memories

Perhaps the most important applications of the magnetic inertia effects lie in the field of data storage.

The use of nondestructive read-out for random access memories of the conventional type is illustrated schematically in Fig. 7. A parallel memory is represented where all the digits of a particular word are read in simultaneously and sensed simultaneously. The read-in process can take place by means of coincident current techniques. A word line corresponding to a particular address is selected by means of the word line selector switch. This switch may utilize vacuum tubes and diodes, transistors, etc. To enter a particular word into the store, the selected word line is pulsed and those digit selection lines corresponding to "ones" in the word are pulsed simultaneously in the well-known manner. To clear a given address the corresponding word line is pulsed with an opposite polarity current in such a way as to reset all the cores in that line.

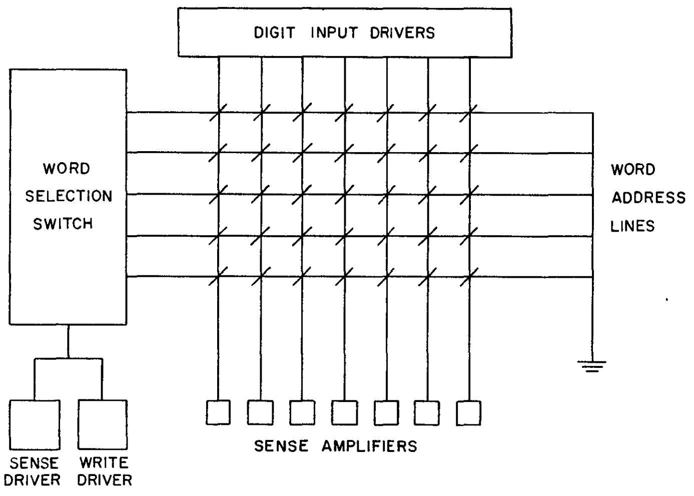


Fig. 7—Parallel-digit memory suited for magnetic inertia operation.

To sense a given address in a nondestructive manner, the corresponding address line is selected by the word selection switch and is pulsed with a limited amplitude pulse of very short duration. Each of the cores on the pulsed line will emit a voltage pulse on the corresponding digit-selection line whose amplitude will be a function of its remanent state. These pulses are sensed by the amplifiers. It should be noted that this well-known configuration has the constructional advantage that the digit-selection lines are used for sensing also. Furthermore the sensing line passing through each core being sensed does not have other half-activated cores in series with it, producing disturbing signals. The signal-noise ratio of the nondestructive sensing effect does not therefore have to be very high in this application.

The advantages of using nondestructive read-out in a random access memory are as follows:

1) Regeneration is unnecessary. This simplifies the logic circuitry required.

2) The fact that a memory interrogation does not have to be followed by a regeneration cycle decreases the effective access time of the memory by a factor whose value depends on the proportion of memory interrogation to memory entries. In the extreme case where all the memory references are interrogations the minimum time between interrogations need be no longer than the sum of the length of the interrogation pulse and the time required to refer to a new address in the memory.

To enter information into a memory by means of the magnetic inertia effects, several modes of operation are possible. One of the most interesting has been named the pulse interlace method and operates as follows.

To switch a particular core in a two-dimensional matrix without disturbing any of the others, anticoincident current pulse trains are sent along the appropriate row and column as shown in Fig. 8 (a). The length and amplitude of each pulse and the number of pulses in each train depend on the material used for the storage elements. In the case of the cores described above, the pulse trains shown in Fig. 8 (b) with 1.4 oersted pulses 0.1 microsecond in length are sufficient.

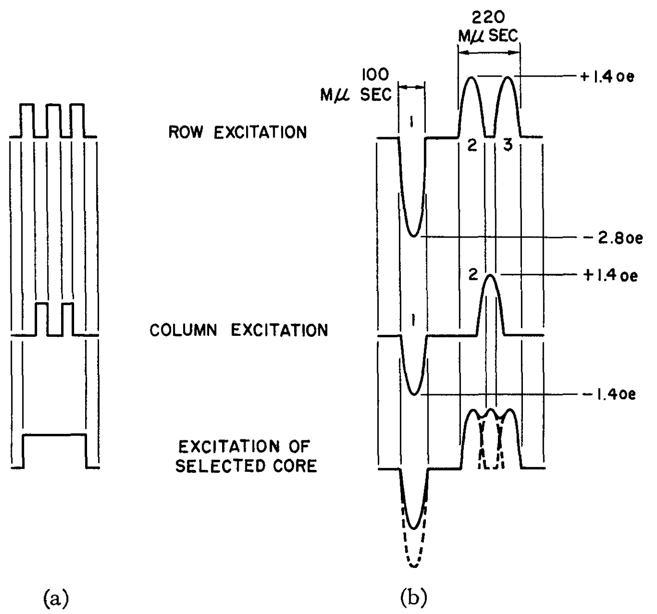


Fig. 8—Pulse interlace waveforms for writing into two-dimensional core matrices.
(a) For use in serial digit arrays. (*e.g.*, Fig. 9).
(b) For use in parallel-digit arrays. (*e.g.*, Fig. 7).

These waveforms are for use in a parallel digit memory of the type shown in Fig. 7. Pulse no. 1 of the row pulse train is used to clear the selected address and is made large enough to perform this purpose even in the absence of column pulse no. 1. Column pulse no. 1 is applied only to those column wires passing through cores which are to be "set" during the read-in portion of the cycle. Its purpose is to prevent column pulse no. 2 from initiating cumulative magnetization changes in cores on unaddressed rows.

The application of row excitation pulses 2 and 3 to a row of reset cores results in a small amount of undesired magnetization change in those cores which have not been switched by the application of a column excitation pulse in the time interval between the two row pulses. This feature is of no consequence in this type of memory structure since every pair of row set pulses is always preceded by a "clear" pulse.

In the pulse interlace method the selected core is excited continuously whereas the nonselected cores are excited discontinuously or not at all. The process of magnetization reversal in the selected core must therefore take place by domain-wall movement.

A method of reversing the magnetization by means of spin rotation makes use of the second threshold effect. To switch a core in a two-dimensional matrix in this way, conventional amplitude coincidence techniques may be used provided that each positive pulse on any one line passing through cores which are not to be switched, is followed or preceded by a negative pulse and vice versa. (These pulses are of half the switching amplitude.) Such a cycle is used in the experimental memory described below.

By making use of the second threshold effect relatively high speeds of operation are possible. Using the type of core described above, it is possible to write into any one core of a matrix by means of two time-coincident 1.4 oersteds 0.1-microsecond pulses preceded by two identical coincident "predisturb" pulses of opposite polarity. The total writing time is approximately ten times as short as the switching time required for the same material using conventional coincident current techniques. The nondestructive read-out procedure which can be used in memories of this type take 0.1 microsecond or less, and is thus at least twenty times as short as the conventional destructive read-out procedure if this has to be followed by re-entry of the sensed information.

Experimental Model of an Inertia Memory

A very small transistorized random-access memory has been built to investigate the use of the second threshold effect for entry of information and of nondestructive read-out for sensing.

The cores described in connection with Fig. 1 were used as the storage element and were arranged in a two-dimensional matrix. The associated control logic is shown in Fig. 9, and enabled the memory locations to be addressed individually for writing, or in sequence for nondestructive read-out.

The core-matrix line-selection circuits use base driven, grounded emitter transistor switches whose collectors are connected in series with the matrix lines through diodes (not shown in the figure). Before pulsing a specific line the emitter base junction of the corresponding switch transistor is forward biased. The application of the matrix-drive pulse thus finds this transistor in a "presaturated" state. This configuration has two advantages.

1) The transistor is not required to undergo a change of state during the application of the drive pulse.
2) The collector base dissipation is a minimum. (Most of the dissipation occurs in the emitter base junction.)

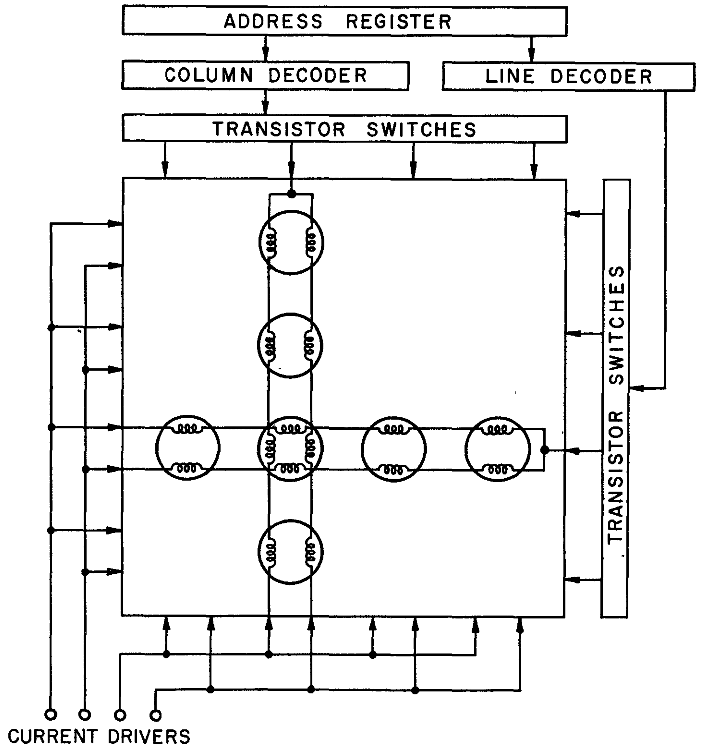


Fig. 9—Two-dimensional inertia memory.

In this way, conventional rf transistors (2N140) are able to switch millimicrosecond pulses of several hundred milliamps at high repetition rates. In the present instance these pulses were generated from commercial vacuum tube equipment.

Writing a "one" into an individual core was accomplished by passing a 0.1-microsecond pulse corresponding to 1.4 oersteds along one of the pair of vertical lines passing through that core as selected by the corresponding transistor switch. Simultaneously, an identical pulse was passed through one of the pair of horizontal lines intersecting the selected core. These coincident pulses had the effect of entering a "zero" into the selected core and of applying a disturb action in the "zero" direction to the other cores on the selected vertical and horizontal lines. To complete the entry of a "one" into the selected core, current pulses were sent along the two previously excited lines. These switched the selected core into the "one" state and cancelled out the disturbance applied to the other cores on the selected vertical and horizontal lines. To enter a "zero" into an individual location, the order of exciting the selected lines was simply reversed.

By setting the address counter and making use of the single cycle facilities of the control equipment, it was verified that a "one" or a "zero" could be entered into any memory location in a single cycle. It was also established that the continuous entry of "ones" or "zeros" into any position of the memory did not disturb the information in other positions. The contents of the memory could be observed on the column lines by cycling the address counters continuously, while applying nondestructive current pulses corresponding to 0.4 oersted to the rows of the matrix.

It is of interest to compare the inertia memory with the evaporated film memory described by Pohm and Rubens.[13] In this memory the elements are switched by the application of two time-coincident field pulses with the resultant field directed at an angle to the easy direction of magnetization. This produces the effect of a transverse as well as of an antiparallel field on the elements being switched. The switching times reported for the evaporated film memory are of the order of 0.5 microsecond. Destructive read-out is used and the sense signals are of the order of 4 millivolts. Since the magnetization in the partially selected elements undergoes reversible rotation, it would appear that nondestructive read-out techniques can be used in a film memory with a corresponding sacrifice in signal strength.

Conclusion

The emphasis in the work done to date has been on the applications of the magnetic inertia effects rather than on a very detailed examination of the effects themselves. An experimental and theoretical study of the wall viscosity should however lead to results of importance to the theory of switching at high fields and may reveal further applications. For example the use of tape-wound toroids where the easy axis of the tape makes an angle with the plane of the toroid should lead to greater speeds of switching for given applied fields at the expense of loss in the effective squareness of the hysteresis loop.

Wall-viscosity effects have been looked for in ferroelectric materials but have not yet been demonstrated in the samples available. Further investigation may reveal their presence.

The construction of memories of large size will necessitate the investigation of techniques for applying the inertia effects to three-dimensional core structures or alternatively to two-dimensional memory arrays driven by magnetic switches.

In the future we hope to be able to present material describing techniques for making use of nondestructive read-out in two-dimensional matrices rather than in one-dimensional matrices as described above.

The work described has been concerned with the use of the magnetic inertia effects in digital devices only. It seems likely however that applications will reveal themselves in allied fields such as that of analog computation.

Acknowledgment

I would like to acknowledge the contributions of W. L. McMillan who played an important part in the work described in this paper and in the design and construction of the two-dimensional memory.

[13] A. V. Pohm and S. V. Rubens, "A compact coincident-current memory," *Proc. East. Joint Comp. Conf.*, New York, N. Y.; December, 1956.

Discussion

G. H. Smith (Autonetics): Would you discuss, briefly, magnetic inertia effects in ferrite cores? Does your research indicate that ferrite cores are practical in the type of memory you described?

Dr. Newhouse: We started work on ferrites after this paper was submitted, and the reason why we concentrated on the metal type cores in the paper is because the switching mechanisms in the metal tapes are similar, and very well known, as compared to the switching mechanisms in ferrites. So it is of more physical interest to concentrate on the metal tape cores. However, it turns out that one can demonstrate all of these effects with pulses on the order of 50 microseconds, particularly the nondestructive read-out effect in conventional square-loop ferrite cores. And the ferrite material in which this has been demonstrated is a material that corresponds closely to the S-1 material, the core material which has been used in most high-speed core memories. The nondestructive read-out effects are, if anything, more pronounced in ferrites than they are in metals. The pulse inhibit effects can also be demonstrated in ferrites, with pulses which are about half the width of the pulses which were used for these metal cores. This, of course, does give a 20 to 1 reduction in cost, from the user's point of view. Also the nondestructive read-out phenomenon can be demonstrated in the aperture ferrite plates.

R. J. Pfaff (IBM): How is a "1" output differentiated from a "0"?

Dr. Newhouse: As shown in Fig. 1 the "1" output is about three times as high as the zero output. With some integration the differences can be made four to one; also these pulses are very constant from one core to the next. As an example, we found that in that experimental memory, we used quarter mil cores by mistake, and we discovered this quite a long time after the memory had been made in the operational.

H. M. Schiller (American Bosch Arma): Please comment on the effect of temperature variations on inertial memories.

Dr. Newhouse: We have not tried any temperature experiments as yet. As I mentioned, if the theoretical mechanisms are half-way correct, we could almost predict it. In general, the Curie temperature of the metals is about 400°C. So you should be able to use the inertia effects at quite high temperatures in expensive metal tape cores. The ferrites, the S-1 ferrite, has a Curie temperature between 100 and 200°C, so in that particular type of ferrite you would presumably have to be careful as you approach the Curie temperature. It is, however, evidently possible that special high-temperature ferrites will be developed in the near future which will have high Curie points.

M. Eisenberg (Thermo Materials, Inc.): How can magnetic domain wall viscosity be defined quantitatively? What would be the effect of temperature on that viscosity?

Dr. Newhouse: It is known that the switching speed of cores varies inversely with the pulse time, so you can draw a straight line by drawing switching times against the applied field. The slope of this line may be expressed in units of centimeters seconds per oersted. And in these practical cases the rough figure of the speed of wall movement is 1000 centimeters per second per oersted. How this has been effected by temperature has been quite carefully investigated by Mena Goodnoff, in a paper, some years ago, in the *Journal of Applied Physics*.

Reliability in Business Systems

HERBERT T. GLANTZ†

Introduction

THROUGHOUT the past few years a great deal of study has been devoted to analyzing the different characteristics and requirements of scientific and commercial data processing systems. Although early general agreement was reached on the fact that such systems *were* different, the exact cause and nature of these variations has not yet been clearly defined. Equipment manufacturers attempted to resolve this difficulty by designing two distinct "lines" of computer models. In some instances these manufacturing distinctions have become blurred with usage, with the result that a Remington Rand Univac is utilized for engineering calculations, while an IBM 704 is applied to payroll preparation. But, in the main, this dichotomy of design and application is being effectively preserved.

Dr. Jay Forrester, the former director of M.I.T. Project Whirlwind, has characterized the chronology of electronic computation as falling into three distinct phases. Beginning in 1945, an intensive amount of research was devoted to investigation of the physical possibility of building electronic digital computers. The early studies of computer logic and circuit design were conducted almost entirely by various engineering universities. By 1950 it was apparent that the basic problems could be solved and that electronic computation would become a reality. Shortly afterwards, the major portion of research activity was shifted to the application of these machines to various problems. The second phase of our history was devoted to computation in the fields of engineering and science as the first large-scale digital computers began to appear in the universities and aircraft companies. In 1955 the emphasis of machine applications research began to switch from engineering to commercial problems. This trend marked our entry into the third and perhaps the most vital phase of development.

As we approached the problem of utilizing electronic equipment for business applications, it became apparent that there were major differences in the requirements of this new area. The early studies devoted to this problem concentrated on seemingly obvious operational variations. Business problems called for vast amounts of input-output data, while scientific problems required lengthy and involved internal calculations. Thus, in short order a convenient categorization grew up: 1) *engineering and scientific applications:* small volume of input-output data; large amounts of complex internal calculations. 2) *commercial applications:* large volumes of input-output data; small amounts of simple internal calculations. Although this fairly arbitrary and sweeping classification proved adequate for early needs of the industry, an increasing sophistication in computer usage has tended to obscure these demarcation lines. It has become increasingly evident that such operational differences do not provide an adequate representation of the two different systems.

† John Diebold and Associates, Inc., New York, N. Y.

If one considers instead, the respective functional purposes of scientific and commercial systems, a striking contrast may be observed. Scientific data processing installations may be regarded as selfcontained systems which function only to satisfy the dictates and requirements of the parent organization. Thus, in a broad sense, all input data originates within the system proper; all computations are determined by the needs of the overall organization; and the timing, quantity, and amount of output information is again dictated by the requirements of the system itself.

Commercial data processing systems, on the other hand, must inevitably exist and perform their functions in constant relationship with the environment of the business world at large. This relationship, which dominates both the design and performance of business systems, is notably evidenced in three ways. The mass of data entering the system originates externally and is to a large extent uncontrolled in format, timing, content, and accuracy. The calculations that are performed are frequently regulated by the rules of outside agencies such as the SEC, ICC, and the Bureau of Internal Revenue. Finally, the daily deadlines that must be satisfied are generally determined by an essentially indifferent environment and are frequently unyielding and seemingly unrealistic.

Accordingly, whereas a scientific data processing system exists and operates to satisfy its own needs, commercial systems must function in large measure to satisfy requirements imposed from outside the system.

The Design of Business Systems

Business systems are growing increasingly more complex as their range of applications expands to cover more demanding and intricate areas. A functioning commercial data processing system includes a variety of information handling components—human, mechanical, and electronic—which are linked together by an overall, communications network. The term "integrated data processing" has gradually come to be accepted as a generic description of the workings of such systems.

The design of these systems has created a striking opportunity for the business world. We are now able to conceive of all routine daily operations of a company being controlled and directed by an automatic system.

Automatic process control is already a familiar occurrence in the chemical and petroleum industries. We are on the verge of applying similar concepts to a number of commercial enterprises. The underlying theme in such systems is automatic control of business operations including a feedback loop for the correction of errors. At the same time, the managing executive is to be provided with information that will allow him to exercise control of over-all company policy on the basis of reliable and timely data.

However, in painting this picture of successful office automation we have overlooked a number of evidently dangerous pitfalls. For as automatic systems exercise a greater amount of control and direction one must place a correspondingly greater emphasis on their operating reliability. Intermittent or periodic failures in such systems can wreak more havoc than is to be gained from long intervals of reliable operation. The increased complexity of these systems introduces more components that are liable to failure and, owing to their varying interrelationships, introduces a greater degree of difficulty in isolating and replacing the faulty elements. Many systems engineers become so enamored with possible accomplishments that they tend to overlook the question of system reliability and the implications of component failure.

As an illustration of the problems that are inherent in the operation of business data-processing systems, we discuss two systems that we have designed for commercial organizations. In both cases we have completely restructured the data-processing operations of the companies involved. In both cases, although substantial dollar savings were accomplished, our primary aim was to have the data processing system provide more effective aid to the sales organization and to top management of the company. In both cases this was effected by centralizing the data processing elements and by utilizing extensive communications networks.

A Stock Brokerage House

"X" is one of the largest stock brokerage houses in the country. Their main volume of business originates on the West Coast where they maintain a number of branch offices. One of the principal requirements of competitive existence in the brokerage field is rapid fulfillment of the customers' orders and requests for sample prices or "quotes." Since most of the clients' trading is done on the floor of the New York Stock Exchange, the company utilizes an extensive private wire system to link the branches with the eastern trading center. It is not unusual for an order to originate in Los Angeles, be flashed to New York and be executed on the Exchange, and be confirmed back to Los Angeles in a matter of two or three minutes.

The system that we have designed for this firm includes an automatic teletype switching center on the West Coast and a data-processing center in New York. The data-processing center utilizes a medium-scale general purpose digital computer as the nucleus of all information processing activities. All branch orders are received by the West Coast message center and are automatically routed on to New York where they are simultaneously sent to the Exchange and to the data-processing center. Execution reports from the Exchange are sent directly to the originating branch and to the data processing center. These orders and execution reports are received on 5-channel teletype tape which is fed directly into the computer. The volume of business handled during the five and one half hour trading day, coupled with over-all speed requirements, necessitate a system capacity of twelve such order-execution pairs per minute.

Basic input is also provided to the system by the branch and New York office reports of daily receipts and disbursements of cash and securities. This extensive input data must be processed each day according to a rigid time table.

In general, all of one day's business activity must be completely recorded and processed before the start of the next trading day. All of this work is subject to the detailed scrutiny of various Exchanges, the SEC, and the auditors.

By utilizing the extensive private wire system, the data-processing center in New York is able to direct the complex daily operational activities of this company. Furthermore, although the company's top management is separated from the data-processing center by 3000 miles, we are able to provide them with a normal supply of reports while at the same time rapidly fulfilling requests for special analysis.

A Transportation Company

"Y" is a medium size company in the transportation industry. The main volume of their work is concerned with cross-country movement of railroad freight cars. The majority of such movements originate in East Coast ports and are destined for western cities. However, a considerable portion of volume consists of overnight movements into such midwestern cities as Chicago, St. Louis, and Cincinnati. A further complication is caused by the frequent necessity of transfer operations, as when Baltimore and Boston shipments are merged in St. Louis before going on to Los Angeles.

A shipping order is prepared by the customer and delivered to the company with each consignment of merchandise. These shipping orders are the basic input to the system and are used to prepare freight bills, railroad manifests, receivables entries, and so forth. A completed freight bill must precede each shipment to its destination and must be on hand at the appropriate transfer points before arrival of the various freight cars. All daily processing activities are subject to ICC regulation and are continually compared by carrier railroads with their own computations. Our client processes an average of 10,000 shipping orders each working day.

The data-processing operations of this company are characterized by a large mass of input data, a requirement for flexible processing schedules that allow for the periodic interruptions of rush movements, and a high volume of printed output accompanied by a moderate amount of punched paper tape output suitable for direct teletype transmission.

Our system design for this company utilizes a medium-scale general purpose digital computer installed in a data-processing center located in Chicago. Owing to the fluctuating nature of data transmission time schedules, the over-all system design is based on a combined usage of direct teletype input and output and an extensive reliance on air mail communication. Original shipping orders, as well as other operational data, are sent to the data-processing center which functions as the controlling element for the entire company. In addition to processing routine daily operating data, the computer is utilized for the formulation of strategic decisions as to selection of optimum freight car routings and carrier-tariff combinations.

Operational Characteristics

The brokerage house and the transportation company are in two vastly different fields of business. Yet the requirements and characteristics of the data-processing systems that have been designed for these companies are strikingly similar. These similarities reveal a great deal about the nature of such business systems: 1) Both companies operate over a wide geographic area utilizing an extensive communications network and a medium-scale computer in a central data-processing installation. 2) In both cases, basic input data is provided by elements that are outside of the system and are essentially disinterested in the workings of the system and the difficulties that are caused by incorrect data. 3) In both cases, input data arrives in a fairly random fashion but must nevertheless be processed upon receipt, since all operations are conducted against fairly intractable time deadlines. 4) In both cases, normal operational schedules must be flexible enough to accommodate the intermittent interruptions of rush jobs. 5) In both cases, internal system processing must conform to rulings of various outside regulatory agencies.

"Real Time" Business Systems

The most important characteristic of both systems, however, is that they are "real time" business systems. The standard definition of a real-time data-processing system is one "whose actions influence the input data that is being received." In the world of business data-processing systems, this definition may be modified to read: "A real-time business system operates on-line with its input data." One such automatic real-time business system is the American Airlines Reservisor system wherein passenger requests for seats are satisfied as they are received with essentially no processing delay.

Until very recently, almost all commercial installation of general purpose computer systems were used in nonreal-time situations and relied extensively on "batch processing" techniques. Examples of areas that are amenable to such techniques form an honor roll of the problems first handled by business systems: payroll, receivables accounting, inventory accounting, insurance and utility billing, and so forth.

The functional nature of a real-time business system requires that essential control and direction of the business be vested in the data-processing system itself. Both of the systems described above receive information concerning the environment or stimuli of the business. In one case these are orders for security trades, and in the other they are shipping orders. The automatic system processes the information contained in these initial messages and issues directions so that successive appropriate measures may be taken. Finally, the data-processing center is notified of the results of these actions and proceeds to issue either corrections or instructions for further measures. All operational data funnels into the processing system and all routine operational directions issue from the system. Further, the formulations of company management are based on analysis prepared by the data-processing system from information that is contained in the records of the system.

An operation of this type sounds delightful to the systems engineer, for we have in large measure eliminated the human element from routine business functions. The automatic system directs and controls daily operations while utilizing the results to provide management with timely reports on which to base long-range policy decisions.

However, the practical businessman views such a system in an entirely different light. The system will reduce direct operating costs; it will provide greater flexibility and efficiency than the present manual system; management will receive information in time to formulate important decisions that direct the company's future operations. But, while accepting the validity of such advantages, the executive also realizes that if this beautiful system should fail, his company will be out of business.

In the final analysis, this factor is the most important characteristic of real-time business systems and provides the greatest single functional difference between on-line and batch-processing systems. If the payroll is late, there will be an unhappy labor force, and if the utility bills are delayed the company will be correspondingly tardy in receiving revenue. But if an on-line system fails, all company operations cease.

Reliability in Real-Time Systems

At the same time, the inherent benefits that may accrue from the utilization of automatic real-time business systems are significantly greater than those provided by the earlier systems. It is just this factor of automatic control of routine operations that provides the appeal of on-line systems. For the effect of a day's activity will be reflected in reports early enough for the executive to exert an effective influence on these same operations. Such real-time data-processing systems are capable of implementing the theory of "management by exception" in actual daily operations.

We thus have a system concept that can provide extensive and valuable benefits to a business organization and can also do irreparable harm in case of failure. Accordingly, we must provide a system design that will reliably ensure against failure while retaining the maximum benefits. Only in this fashion can we hope to implement this significant step foreward in the application of automatic data processing.

In the design of real-time business systems we have adopted a concept that is based on the assumption that all components of a system are liable to failure, but that over-all operations must not fail. One of the simplest and most effective means of ensuring system reliability is through the use of redundancy. This technique is utilized by the human brain and nervous system, and the analogy is a natural one since the data-processing system functions in a similar fashion for the business organism. However, in view of the economic realities of commercial life, it would appear more practicable to approach the problem in a slightly different manner.

The first requisite of our system is a high quality of design and performance in each individual element. Practically, this implies that only proven components can be incorporated in the system. As newer computers are developed, they should be "broken in" on the batch-processing problems. If that is not feasible, one must insist on a substantial period of rigorous testing before accepting such equipment.

Constant preventive maintenance of all equipment is required. A number of engineering installations of medium-scale computers adopted the practice of eliminating preventive maintenance periods. These organizations have found it more economical to run the computer until it breaks down before calling for the engineers. It seems obvious, however, that this practice could not be tolerated in our system.

The mere availability of "backup" equipment does not provide insurance. Since the computer in our system functions as the nerve center of a complex communications network it will generally be impossible to transfer operations to another machine at a distant location without severely disrupting the information flow. However, commercial teletype lines and neighboring printer units can be utilized as temporary replacements for certain system components.

A fundamental requirement in our system design is the provision of means for human intervention. Intermittent failures in system components are to be expected and their functions must be performed during the interval necessary to correct and replace the mechanical faults. At present, an intelligent human being is the surest substitute and the safest means of providing against the varying and complex difficulties that can arise. Accordingly, it is necessary to provide the mechanical system with a capable and trained operating staff. This staff must be thoroughly aware of precisely what the system is accomplishing so that it can immediately introduce the correct remedial steps in case of failure. We have found it wise to run periodic "alerts" in which a variety of failures are simulated and the proper countermeasures are promptly initiated.

Application of the principles of automatic control to the operation of business organizations presents an exciting opportunity. Commercial data-processing systems designed to utilize these techniques will assume responsibility for the direction and control of all routine daily functions. The utilization of comprehensive communications networks will enable a central data-processing unit to exercise effective supervision over operations of the entire organization. The incorporation of feedback loops in the system will provide a means of notifying the data-processing center of the results of earlier activities. Management reports will be abstracted from the flow of daily information, thus providing the executive with an accurate analysis of the current situation.

As such real-time business systems assume more overall responsibility and control, an even greater stress must be placed on the reliability of their operation. For failure in these systems implies not just delay and annoyance but almost complete cessation of activity. It is imperative that extreme care be exercised in the design and operation of on-line systems so that all component failures are provided against. Only in this fashion will we be adequately assured that the possibility of total failure has been eliminated and that operations can be safely entrusted to the system.

Discussion

C. K. Budd (U. S. Army Signal Corps): What has been your experience on volume of teletypewriter transmission and number or percentage of errors? How are errors detected and corrected?

Mr. Glantz: Our experience has been that people always claim that you cannot rely on teletype transmission, and yet when actual studies are made they invariably turn out to be operator error rather than machine error. We have found that in terms of actual equipment error the percentage rate is well under one per cent. Our freight car company is transmitting something like ten thousand shipment orders per day, of which perhaps one-third to a half on a normal day will go over the wires, and of which somewhat over half of the resulting process information will be transferred over wires. We have found that one of the advantages of an on-line system such as this is that if the computer itself puts out the paper tape which will go over the teletype system, it eliminates the possibility of human error and will have a great deal more of reliability in the system.

From the floor: Is there always a paper tape between a medium speed computer and communication lines?

Mr. Glantz: In each of the systems that we have designed paper tape is punched out on the teletype system, fed into a switch complex, and then into the computer, which will punch out paper going back as well.

This is obviously not necessary, particularly on the Remington Rand Reverse System which is under design now. We will have direct communication between the agencies and the computer which acts as a central processing unit. We feel that for the moment it is of great value to have paper tape physically punched out.

From the floor: What proportion of the over-all cost of this system does the computer represent?

Mr. Glantz: About two-thirds of the over-all hardware. If personnel is included, this fraction drops considerably.

From the floor: In the computer that you used in the brokerage operation, can you always be sure that it has broken down? Can you be sure that one day is long enough to make your diagnosis every time?

Mr. Glantz: No, you cannot be sure.

From the floor: In the freight car loading example, the computer center was in Chicago. Do you have a master file that you run through on a random access basis in that particular system, or do you have segregated parts of a master file you refer to specifically for each type of processing run?

Mr. Glantz: The closest thing we have to a master file is a record of all freight shipments. Now, these are normally broken down in the file by day on which they occurred and by cities from which they came, normally on the East Coast. The location of Chicago was picked out simply as a matter of function convenience. Since more shipments go to the East or West Coast, we thought that if the shipping order was picked up on the East Coast and air mailed to a teletype in Chicago, we could have most of the processing done when the shipment came through Chicago and the mid-West.

On Prediction of System Performance from Information on Component Performance

JOAN R. ROSENBLATT†

Introduction

THE PURPOSE of this paper is to propose some building blocks for a systematic approach to prediction of the performance or reliability of complex equipment from information on component performance. Particular attention is given to the use of information on ways in which components are believed or known to be interdependent, functionally as well as structurally.

The reliability of a system is defined as the probability P that its satisfactory operating life under stated conditions is not less than a specified time T. This probability can, in principle, be estimated directly from life tests of a large number of systems. In practice, however, this is usually undesirable or impossible. In particular, it would often be desirable to make at least approximate estimates of P before a complete system has actually been assembled.

This paper presents a systematic approach for examining engineering design information, data on component performance, and information on conditions of use, in order that statistical techniques may be used to obtain an estimate of P.

It is assumed that there may be obtained a collection of statements such as the following, describing the conditions on component behavior which permit satisfactory operation of the system:

> "The system will operate only if the performance of component A is satisfactory (in a precisely defined way.)"
>
> "The system will operate only if properties X_B and Y_C of components B and C satisfy a specified relation (such as $X_B+Y_C\geq$constant)."
>
> "The system will operate only if at least one of components D and E has not failed."

It is assumed that this collection of statements contains all the essential relations determining *the dependence of system performance on component performance*. Then P is equal to the probability that all of these conditions will be met for the length of time T.

Additional information of the following kind may be used:

> "Components A, B, C will all fail at once if certain unusually extreme atmospheric conditions prevail."
>
> "As the performance of component A deteriorates, the load on component B is decreased and deterioration of component B is slowed down."

These statements illustrate interrelations among components in their response to the conditions under which the system is operated. Information of this kind is discussed below in the section on *interdependence of component failures*.

It is shown in *some simple hypothetical examples* how this additional information on interdependence among components may be introduced into a mathematical expression for P.

In the context of this approach to the analysis of system performance, the following problems are considered: 1) estimating and giving *confidence intervals*[1]

† Natl. Bureau of Standards, Washington, D. C.

[1] See, *e.g.*, W. J. Dixon and F. J. Massey, Jr., "Introduction to Statistical Analysis," McGraw-Hill Book Co., Inc., New York, N. Y.; 1951.

for P from data obtained in tests of components and subassemblies; 2) using the mathematical expression for P, together with information about costs, to aid in making decisions about design changes such as the addition of redundant elements, and 3) using the mathematical expression for P to simulate the effect of component specifications on system performance.

Preliminary Remarks; The Problem of Detail

It is important to state what sort of composite entity is considered in this paper to be a "system," and to indicate how the "components" of a system are viewed. A system is understood to be any equipment, subsystem, or device composed of subsidiary parts, whose joint performance determines the performance of the system in respect to one or more properties. The components of a system are those parts, at a selected level of detail, whose performance is to be related to system performance.

In particular, it is suggested that a useful approach to predictions of system performance is obtained by thinking of a complex system as organized in *levels*. Thus, the probability of successful performance of a complex system may be considered first as determined by the behavior of a relatively small group of major subsystems. The latter may then be analyzed in turn. This procedure should lead to efficient concentration of effort toward obtaining detailed information about subsystems which have the most important effect on system performance. With this approach, moreover, information is organized in a way which permits selection of the level of detail appropriate to the kind of question which is to be asked.

When the "system" under consideration is in fact a subsystem of a larger system, it will not always be sufficient to summarize performance by the probability of successful operation. For subsystems, the "measurement" of reliability requires a somewhat different approach—as will be seen in the discussion of the hypothetical examples given below.

Discussions of Interdependent Components

It has been observed frequently that assessments of system reliability based on the assumption that component failures occur independently of one another are approximate and usually excessively conservative. Several authors have discussed possibilities for representing the dependence of system performance on the performance of interdependent components. Some of these discussions are briefly noted here.

Statistical analysis of circuit performance has been discussed by Benner and Meredith,[2] and Meltzer.[3] Performance characteristics are given approximately as functions of circuit elements, so data on behavior (means, variances, and correlations) of the circuit elements can be used to predict behavior of the circuit.

If it is not possible on theoretical grounds to state the relation of system performance to component behavior, it may be possible to use multiple linear regression analysis to estimate the relationship, at least for narrow ranges of values of component characteristics. This approach has been discussed by Brown[4] and Bear.[5] If the estimated relationship (a linear equation) appears to give a satisfactory representation of the dependence of system performance on component characteristics, then it may be used to predict the effect on system performance of variability in component performance.

The details of these procedures will not be discussed in this paper. They are available for obtaining functional relationships between system performance and component performance in situations where their methods are applicable.

Another approach to making use of information about interdependence among components has been suggested by Elmaghraby,[6] who considers a representation of the effect of component failures which do not directly cause system failure but increase the probability of failure of other components.

The first two main sections of this paper are devoted to discussions of the use of information relating system performance to the performance of interdependent components.

Dependence of System Performance On Component Performance

The first kind of information which is needed for relating system performance to component performance is illustrated by statements such as those discussed below. These are conditions on component behavior which permit satisfactory operation of the system. They are determined by the nature of the components and the conditions of their application in the system. These conditions are distinguished from sources of failure, which are considered later.

The system will operate only if the performance of component A is satisfactory. Some components (or subsystems) have a direct and "independent" effect on system performance, in that the system cannot operate properly if these components fail to give satisfactory performance, no matter what be the performance of other components. The dependence of system performance on these components is then given by a definition as precise as possible of satisfactory performance for each of them.

[2] A. H. Benner and B. E. Meredith, "Designing Reliability into Electronic Circuits," Symposium of National Electronics Conference; October, 1954.

[3] S. A. Meltzer, "Designing for reliability," IRE Trans., vol. PGRQC-8, pp. 36–43; September, 1956.

[4] H. B. Brown, "The role of specifications in predicting equipment performance," *1956 Proc. Second National Symposium on Quality Control and Reliability in Electronics*, pp. 133–148.

[5] J. C. Bear, "Elements of Reliability Prediction," Arinc Monograph No. 4, Aeronautical Radio, Inc.; October 1, 1956.

[6] S. E. Elmaghraby, "A Generalization in the Calculation of Equipment Reliability," Cornell Univ., School of Elec. Eng., Res. Rep. EE 314; November 15, 1956.

It should be emphasized that a component whose effect on system performance is "independent" (in the sense that the definition of satisfactory performance for the component does not involve conditions on the performance of other components) may, nevertheless, ***not be independent*** of other components when causes of failure are considered. For example, the front and rear hand-brakes of an English bicycle operate independently, and the bicycle may be said to operate satisfactorily only if both are working; but a slippery street affects the performance of both brakes. In the long run, the proportion of successful operations of the braking system will depend on the frequency of rainy days as well as on the frequency of hand-brake breakdowns.

Some components of this type have a simple go-no-go relation to system performance: if a crucial solder joint is loose, the system just cannot work. Many components, however, have a more complicated relation to system performance: some set of numerically measured characteristics of the component must remain within (known) stated bounds if the system is to operate satisfactorily.

For components having independent direct effect on system performance, the dependence of system performance on component performance may be represented on a zero-one basis. Each component either meets the stated condition or not.

The value of the system output variable x_0 is determined by a known function of characteristics of a set of n components $C_1, C_2, \cdots, C_n$. Suppose $y_1, \cdots, y_n$ are respectively the relevant characteristics of $C_1, \cdots, C_n$, and that it is known that $x_0 = f(y_1, \cdots, y_n)$ where the form of the function $f(y_1, \cdots, y_n)$ is known. If the value of x_0 must be in a stated range, then the values of $y_1, \cdots, y_n$ must be restricted so that $f(y_1, \cdots, y_n)$ is in this range. In this situation, the components $C_1, \cdots, C_n$ are evidently interdependent; a given value of y_1 may be satisfactory when associated with one set of values for $y_2, \cdots, y_n$ but unsatisfactory when associated with some other set. The dependence of system performance on performance of components $C_1, \cdots, C_n$ is given by $f(y_1, \cdots, y_n)$, and cannot in general be stated in terms of individual conditions on the characteristics $y_1, \cdots, y_n$.

The system will operate only if properties y_1 and y_2 of components C_1 and C_2 satisfy a specified relation. An example of such a relation is the condition that $(y_1 + y_2)$ must be greater than a specified number. The situation in this case is essentially the same as that of the preceding case.

The system output variable x_0 depends on characteristics $y_1, \cdots, y_n$ of components $C_1, \cdots, C_n$; but the theoretical relation does not agree adequately with relations observed in practice. This situation prevails, for example, when the best available theoretical analysis of a system is based on ideal physical properties of the elements of the system, while realizations of the elements cannot have these ideal properties. (Thus, to some degree, this situation always prevails.) If it has been discovered empirically that the theoretical analysis leads to inadequate predictions of system operation, it may have been discovered also what kinds of discrepancies occur. The representation of system performance as a function of component performance must be obtained by "educated guessing." Once a mathematical expression for the reliability of the whole system has been constructed, then it can be determined by varying the conjectured relationship whether or not the accuracy of the guess will have an important effect on the accuracy of the over-all prediction. In some situations, it may be feasible to perform an experiment to estimate the empirical relation of a set of component characteristics to system performance.

The probability distribution of the system output variable x_0 depends on characteristics $y_1, \cdots, y_n$ of components $C_1, \cdots, C_n$. This statement distinguishes a case in which the relation of system performance to component performance is statistical. In the previous cases, the probability distribution of x_0 would depend on the probability distributions of $y_1, \cdots, y_n$; this remains true, but it is supposed further that fixed values of $y_1, \cdots, y_n$ determine not the exact value of x_0 but, for example, the average value to be expected for x_0. If the form of the distribution of x_0 is known—*e.g.*, if x_0 has a Gaussian distribution with its mean and variance (mean-square deviation from mean) determined by known functions of the values $y_1, \cdots, y_n$—then the statistical relationship of x_0 to $y_1, \cdots, y_n$ is specified. This case differs from the other kinds of relationships discussed, in that there may be no values of $y_1, \cdots, y_n$ for which it is certain that the value of x_0 will be satisfactory. It will be seen, nevertheless, that it is possible (though not easy) to make predictions about the probability that x_0 will have a satisfactory value.

The system will operate only if at least one of components A and B gives satisfactory performance. This is the situation if a system contains duplicate or redundant components. When the duplicate component is present on a stand-by basis to be used only if the first one fails, there may be a third component involved which detects the failure and puts the stand-by unit into operation. The representation of this kind of relationship and its effect on reliability has been discussed by Luebbert.[7]

The Composition of Conditions

Suppose now, that all essential conditions on component behavior which determine satisfactory system performance are effectively accounted for. That is, suppose a set of components or subsystems have been listed together with the relation of system performance to the characteristics of each. Abstractly, the situation at this point may be illustrated by the following example.

[7] W. F. Luebbert, "Principles and Concepts of Reliability for Electronic Equipment and Systems, Part II: Simple Models for Failure of Complex Equipment," Stanford Univ., Electronics Res. Lab., Tech. Rep. No. 91; August 18, 1955.

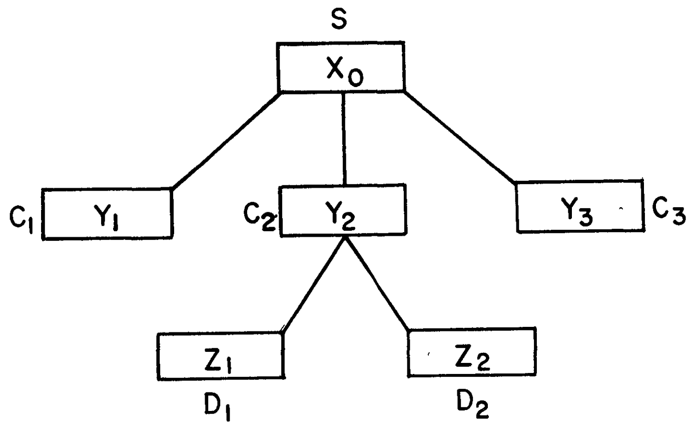


Fig. 1—Relation of components in a simple system S.

Fig. 1 is a diagram of a simple system S whose satisfactory performance (apart from catastrophic failure) is defined in terms of the output variable x_0. The system S is composed of three subsystems C_1,C_2,C_3 with outputs y_1,y_2,y_3, respectively. The subsystem C_2 is further analyzed in terms of components D_1 and D_2. The system S is said to be giving satisfactory operation as long as

$$a \leq x_0 \leq b,$$

where a and b are given numbers. The dependence of system performance on the performance of the subsystems C_1,C_2,C_3 is given (say) by the following statements.

1) If $y_1<2$, the system S fails.
2) If $y_1+y_2>10$, the system S fails.
3) $x_0=Ke^{y_3}$.
4) If any one of C_1,C_2,C_3 has a catastrophic failure, the system S fails.

Thus the probability P that S will give satisfactory performance is given by

$$\text{Prob. } \{C_1 \text{ does not fail } \textit{and} \\ C_2 \text{ does not fail } \textit{and} \\ C_3 \text{ does not fail } \textit{and} \\ y_1 \geq 2 \textit{ and} \\ y_1+y_2 \leq 10 \textit{ and } \log(a/K) \leq y_3 \leq \log(b/K)\}.$$

It is clear that the conditions connected by "and" in this expression are not all independent. One kind of dependence among components is explicitly stated in the condition "$y_1+y_2\leq 10$." Another kind of dependence is suggested by the distinction between the conditions "C_1 does not fail" (*i.e.*, no catastrophic failure) and "$y_1\geq 2$" (*i.e.*, no wearout failure—that is to say, the output of C_1 is at a satisfactory level). Further consideration of this sort of interdependence among components is deferred until the next section of this paper.

Continuing with the analysis of the system of Fig. 1, consider now the subsystem C_2 with two components D_1 and D_2. Suppose the dependence of C_2 on the performance of D_1,D_2, and their output variables z_1,z_2 is given as follows.

1) If both D_1 and D_2 fail, the subsystem C_2 fails.
2) The value of y_2 is uniformly distributed in the interval $z_0\pm 1$, where z_0 is the larger of z_1,z_2.

Restating condition 2), it is supposed that the value of y_2 is not precisely determined by z_0, but will be within ± 1 of z_0 with any value in this range being as likely as any other. If it is desired to expand the expression for P given above, then the condition "C_2 does not fail" is replaced by "D_1 does not fail *or* D_2 does not fail." Furthermore, letting $U(z_0)$ denote a random variable uniformly distributed on the interval $z_0\pm 1$, the condition "$y_1+y_2\leq 10$" is replaced by "$y_1+U(z_0)\leq 10$."

This illustrative example will be considered further below, to illustrate the incorporation of additional information about interrelations among the components of S. The next main section of the paper deals with interdependence among components with respect to their modes of failure.

Interdependence of Component Failures

Additional information which will be useful for assessing the relation of system performance to the performance of interdependent components is illustrated by statements such as those discussed in this section. These statements are typical of some possible interrelationships among components, arising from the nature of the application of the components in the system, and from the response of component performance to the conditions under which the system is used.

Components A, B, and C will all fail at once if certain unusually extreme atmospheric conditions prevail; under normal conditions, they suffer other kinds of catastrophic failures or wearout failures independently of one another. This situation may be described by the term "conditional independence." Suppose, for instance, that a system is used in such a way that it is subjected to some kind of severe shock from time to time, and has been designed to withstand such shocks unless they are unusually severe. Prediction of system performance may be more realistic if this particular kind of failure is treated separately. The calculations which are appropriate in the conditional independence situation are illustrated by a simple numerical example.

Consider two components A and B. Suppose that each one can exhibit three kinds of performance: satisfactory, unsatisfactory because of wearout, or failure, due to unusual shock. Suppose further that it has been established that the probabilities of each of these have been determined to be as follows, where $p_a+q_a+r=1$, $p_b+q_b+r=1$.

Component	Satis.	Unsatis.	Shock
A	$p_a=0.980$	$q_a=0.010$	$r=0.010$
B	$p_b=0.985$	$q_b=0.005$	$r=0.010$

The probability r of shock failure is the same for each component, since it depends not on the nature of the particular component but on the occurrence of an unusual ambient condition. One way to look at the joint performance of components A and B is to consider the joint probability distribution, which gives for every pair (performance of A, performance of B) the probability of its occurrence. The following table gives the joint distribution for components A and B, assuming conditional independence.

Example 1: Joint Probabilities of Performance

Performance of A	Performance of B		
	Satis. (p_b)	Unsatis. (q_b)	Shock (r)
Satis. (p_a)	$\frac{p_ap_b}{1-r}=0.9751$	$\frac{p_aq_b}{1-r}=0.0049$	0
Unsatis. (q_a)	$\frac{q_ap_b}{1-r}=0.0099$	$\frac{q_aq_b}{1-r}=0.0001$	0
Shock (r)	0	0	$r=0.0100$

It is seen that the quantities in each row add up to the probability of the corresponding performance of component A, and that the columns add up to the corresponding probabilities for component B. The sum of all the entries is unity. The probabilities may be derived from the conditional independence assumption by the rules for calculating with conditional probabilities.[8] Let $P(U|V)$ denote "the probability of U on condition that V is true." Then, for example,

$$P(A \text{ satis. } and\ B \text{ satis.})$$
$$= P(A \text{ satis. } and\ B \text{ satis.} \mid \text{shock occurs}) \times P(\text{shock occurs})$$
$$+ P(A \text{ satis. } and\ B \text{ satis.} \mid \text{no shock}) \times P(\text{no shock}).$$

Now the first conditional probability is obviously zero; if an unusual shock occurs, both A and B fail. Thus, only the second term must be evaluated. Since A and B are assumed to be independent under normal conditions (no shock), the second conditional probability may be written as the product of two conditional probabilities,

$$P(A \text{ satis. } and\ B \text{ satis.} \mid \text{no shock})$$
$$= P(A \text{ satis.} \mid \text{no shock}) \times P(B \text{ satis.} \mid \text{no shock}).$$

Evaluating these conditions from the information given about the probabilities for the individual components, it follows that

$$P(A \text{ satis.} \mid \text{no shock}) = p_a/(1-r),$$
$$P(B \text{ satis.} \mid \text{no shock}) = p_b/(1-r).$$

Finally, the probability of no shock is $(1-r)$. Thus,

$$P(A \text{ satis. } and\ B \text{ satis.}) = \frac{p_a}{1-r}\cdot\frac{p_b}{1-r}\cdot(1-r) = \frac{p_ap_b}{1-r}.$$

[8] W. Feller, "An Introduction to Probability and Its Applications," John Wiley and Sons, New York, N. Y.; 1950.

The prediction of system reliability obtained from a joint distribution of this kind will be somewhat different from a prediction made on the assumption that the performance of the two components is unconditionally independent. To illustrate this, consider two cases. Case I: Both A and B must give satisfactory performance in order that the system operate correctly. Case II: At least one of A and B must give satisfactory performance. If it is assumed that the two components are unconditionally independent, then the probability that the joint performance of A and B will be satisfactory is calculated from the given probabilities p_a and p_b by well-known methods.[7]

Case I: $P(A \text{ satis. } and\ B \text{ satis}) = p_ap_b$.
Case II: $P(A \text{ satis. } or\ B \text{ satis.}) = p_a + p_b - p_ap_b$.

Under the conditional independence situation, the Case I probability may be read directly from the joint distribution table. The Case II probability may be calculated from the table by the usual formula:

$$\begin{aligned} P(A \text{ satis. } or\ B \text{ satis.}) &= P(A \text{ satis.}) + P(B \text{ satis.}) \\ &\quad - P(A \text{ satis. } and\ B \text{ satis.}) \\ &= p_a + p_b - p_ap_b/(1-r). \end{aligned}$$

The numerical comparison is as follows.

Probability That Joint Performance of A, B Is Satisfactory

	Independence	Conditional Independence
Case I (series) *A and B* satis.	0.9653	0.9751
Case II (parallel) *A or B* satis.	0.9997	0.9899

It is true in general that if the present kind of dependence situation prevails, reliability predictions based on the independence assumption will be overconservative for components connected in series, but overoptimistic for components connected in parallel. For Case I, the differences between the two predictions will be greater if there are more than two components. For Case II (parallel), increasing the number of components makes the probability of satisfactory joint performance become close to unity very rapidly under unconditional independence; while under conditional independence the upper bound for the probability of satisfactory joint performance is $(1-r)$.

If a system contains n identical components, connected in series, and the probability of satisfactory performance for each component is p while the probability of a "shock failure" is r, then under the conditional independence assumption the probability that the whole set of n components will give satisfactory performance is

$$p^n/(1-r)^{n-1}.$$

Thus, suppose an equipment such as a computer contains 1000 components each of whose failure probabilities $(1-p)$ is 1/10,000. And suppose the "shock failures" which affect all components simultaneously are about 10 per cent of these, *i.e.*, $r=10^{-5}$. The predicted frequency of failures calculated by the product rule would be

$$1 - p^n = 0.095$$

while the prediction based on the conditional independence situation would be

$$1 - p^n/(1 - r)^{n-1} = 0.086.$$

As the performance of component A deteriorates, the load on component B is decreased and degradation of performance of B is slowed down. For example, suppose the following extreme situation prevails. If component A wears out, then component B will surely continue to give satisfactory performance; and vice versa. (For simplicity, it is assumed that catastrophic failures are impossible.) This situation is a case of "negative dependence." Let the probabilities for A and B be given as follows, where $p_a+q_a=1$, $p_b+q_b=1$.

Component	Satis.	Unsatis.
A	$p_a=0.98$	$q_a=0.02$
B	$p_b=0.97$	$q_b=0.03$

Under the assumption of negative dependence described above, the joint probability distribution for the performance of components A and B is given by the following.

Example 2: Joint Probabilities of Performance

Performance of A	Performance of B	
	Satis. (p_b)	Unsatis. (q_b)
Satis. (p_a)	$p_a+p_b-1=0.95$	$q_b=0.03$
Unsatis. (q_a)	$q_a=0.02$	0

These probabilities are derived from the assumptions that

$$P(A \text{ satis.} \mid B \text{ unsatis.}) = 1$$

$$P(B \text{ satis.} \mid A \text{ unsatis.}) = 1.$$

Thus, for instance,

$$P(A \text{ satis. } and\ B \text{ unsatis.})$$

$$= P(A \text{ satis.} \mid B \text{ unsatis.}) \times P(B \text{ unsatis.})$$

$$= P(B \text{ unsatis.}) = q_b.$$

To see the effect of this kind of dependence on reliability prediction, consider again the two types of possible application of components A and B in a system: Case I (series) and Case II (parallel). The following shows the difference between predictions made on the basis of the independence assumption and the predictions appropriate in the present type of negative dependence situation.

Probability that Joint Behavior of A, B Is Satisfactory

	Independence	Negative Dependence
Case I (series) *A and B* satis.	0.9506	0.9500
Case II (parallel) *A or B* satis.	0.9994	1.0000

Observe that in this situation, the independence assumption tends to give an overoptimistic prediction in Case I, and to underestimate the effectiveness of redundant components (Case II). This is the opposite of the previous (conditional independence) situation, where the dependence between the two components was positive (shock failures occurred simultaneously in both components).

Failure reports from the field have indicated that every time component A fails, component B fails also. This statement illustrates a particular kind of positive dependence which may be called "chain dependence." Failure of component A always leads to failure of component B, while component B may be subject to additional types of failure.

Example 3: Joint Probabilities of Performance

Performance of A	Performance of B	
	Satis. (p_b)	Unsatis. (q_b)
Satis. (p_a)	p_b	q_b-q_a
Unsatis. (q_a)	0	q_a

As usual, $p_a+q_a=1$, $p_b+q_b=1$. Once the zero has been inserted in the cell corresponding to the type of joint performance which is assumed to be impossible, the remaining entries shown above are determined by the requirement that the rows and columns each add up to the appropriate individual probabilities. The following computation provides a comparison between the assumption of independence and the assumption of chain dependence as given in the table. Let $q_a=0.01$, $q_b=0.02$.

Probability of Successful Joint Performance (*A* and *B* Satis.)

Independence:	$p_a p_b=0.9702$
Chain Dependence:	$p_b=0.9800$

The three examples given in this section of the paper were intended to illustrate some of the possibilities for obtaining mathematical expressions which incorporate information about the occurrence of failures. In the next section, some additional assumptions are made about the system S of Fig. 1, and the calculations for that example are continued.

Some Simple Hypothetical Examples

First Illustrative System, System S

Consider again the system S of Fig. 1, and suppose that it is desired to make use of some additional information about interdependence among failures of the components of S.

Shock Failure: Analysis of component properties and of the proposed conditions of use for the system S have indicated that all components will fail simultaneously if the system is exposed to a certain extreme atmospheric condition. The probability of this occurrence is r, and joint occurrence of this shock situation and other failures is impossible. Then (*cf.*, Example 1 of the preceding section) all remaining calculations are understood to be valid on condition that no shock has occurred. The general rule is

$$\Pr(U \text{ and } V) = \Pr(U \mid V) \times \Pr(V).$$

Accordingly, the probability P of successful system performance is obtained from the probability of successful system performance on condition that no shock has occurred, by multiplying the latter by $(1-r)$—the probability that no shock occurs.

Component C_2: Recall that this component is in fact a subsystem containing two components D_1 and D_2. Suppose that (on condition no shock has occurred) the components D_1 and D_2 are independent and have the same performance probabilities. Let q_D be the probability of a catastrophic failure (other than shock failure) for D_1 or D_2 and let the probability distributions for the output variables z_1, z_2 be $f_D(z_1)$, $f_D(z_2)$, respectively (when no catastrophic failure has occurred).

Suppose that, except for shock failure, failures of component C_2 occur independently of the performance of components C_1 and C_3.

Components C_1 and C_3: Suppose component C_3 is believed to be chain-dependent on component C_1 (*cf.*, Example 3 of the preceding section). In particular, let q_{C_1} and q_{C_3} be the respective probabilities of catastrophic failure (on condition of no shock), and suppose the joint probabilities of catastrophic failure are as follows.

JOINT PROBABILITIES OF CATASTROPHIC FAILURE

Component C_1	Component C_3	
	No Failure $(1-q_{C_3})$	Failure (q_{C_3})
No failure $(1-q_{C_1})$	$1-q_{C_3}$	$q_{C_3}-q_{C_1}$
Failure (q_{C_1})	0	q_{C_1}

Suppose further that if no catastrophic failure has occurred C_1 and C_3 are independent. That is, the values of the output variables y_1 of C_1 and y_3 of C_3 have independent probability distributions $f_{C_1}(y_1)$ and $f_{C_3}(y_3)$ when no catastrophic failure has occurred to either one.

Expression for P: Recall now the expression for the probability P that the system S will give satisfactory performance. P was expressed as the probability of joint occurrence of several events. With the assumptions now given, P can be expanded by repeated applications of the rule

$$\Pr(U \text{ and } V \mid W) = \Pr(U \mid V \text{ and } W) \times \Pr(V \mid W).$$

The result is

$$\begin{aligned} P = {} & (1-r) \times (1-q_D)^2 \times (1-q_{C_3}) \\ & \times \Pr(y_1 \geq 2 \text{ and } y_1 + U(z_0) \leq 10 \mid Q) \\ & \times \Pr\left(\log\frac{a}{K} \leq y_3 \leq \log\frac{b}{K} \mid Q\right) \\ & + 2(1-r) \times q_D(1-q_D) \times (1-q_{C_3}) \\ & \times \Pr(y_1 \geq 2 \text{ and } y_1 + U(z_1) \leq 10 \mid Q^*) \\ & \times \Pr\left(\log\frac{a}{K} \leq y_3 \leq \log\frac{b}{K} \mid Q^*\right) \end{aligned}$$

where Q stands for the condition "C_1 does not fail *and* neither D_1 nor D_2 fails *and* C_3 does not fail;" Q^* denotes the condition "C_1 does not fail *and* exactly one of D_1, D_2 fails *and* C_3 does not fail."

Now, $\Pr(\log a/K \leq y_3 \leq \log b/K \mid Q)$ may be calculated from the probability distribution $f_{C_3}(y_3)$—and is the same whether the condition is Q or Q^*. Under condition Q, the probability of "$y_1 \geq 2$ *and* $y_1 + U(z_0) \leq 10$" can in principle be calculated from the probability distributions $f_{C_1}(y_1)$, $f_D(z_1)$, and $f_D(z_2)$. The distribution of $z_0 = \max(z_1, z_2)$ is determined first; from this, the distribution of $U(z_0)$ is obtained. Under condition Q^*, exactly one of components D_1, D_2 has not failed, so that z_0 is equal to z_1 (say). The distribution of z_0 is then simply $f_D(z_1)$, and the distribution of $U(z_1)$ is obtained from it. The second term of the expression for P is multiplied by 2 because the condition Q^* can occur in two ways (D_1 fails or D_2 fails) while the probabilities are the same for both.

It may be seen that the expression for P has now been expanded into a form which can be calculated when the following data are available: 1) estimates of the probability distributions for output variables y_1, y_3, z_1, z_2 on condition that no catastrophic failure occurs; 2) an estimate of the probability of shock, r; 3) an estimate of q_D, the probability of catastrophic failure of D_1 (or D_2) on condition that no shock has occurred, and 4) an estimate of q_{C_3}, the probability of catastrophic failure of C_3 on condition no shock has occurred.

Second Illustrative System, System R

The preceding example was constructed to illustrate the possibility of using various types of information together. The present example has less variety, with a larger number of components. Suppose a system R is composed of two identical major subsystems R_1 and R_2 with output variables x_1 and x_2. The output x of R is satisfactory if $x \geq a$, where a is a fixed number; and x equals the larger of x_1, x_2. The subsystem R_1 is composed of a power source which is either working or not; and 100 identical components, each one of which is either working correctly, or not. The output x_1 of R_1 is a function $x_1 = f(n_1)$ of the number n_1 of components of R_1 which are working. Similarly, $x_2 = f(n_2)$.

There is a possibility of shock failure (propability r) which would affect all components at once. In the absence of shock failure, R_1 and R_2 are negatively dependent with respect to the occurrence of power failures (say there is always exactly one auxiliary power source available). The probability that a power source fails is q. In the absence of shock or power failures, all components of R_1 and R_2 are independent and each has (conditional) probability p of working.

With this information, the distribution of $x_1=f(n_1)$ may be calculated from the binomial probability distribution[8] which governs the performance of the 100 components of R_1 in the absence of shock or power failure. This calculation requires only an estimate of p [and of course knowledge of the form of the function $f(n_1)$]. Given the (identical) distributions of x_1 and x_2, the distribution of $x=\max(x_1, x_2)$ on condition of no shock or power failure can be calculated. An expression for P is then obtained as follows.

$$\begin{aligned} P = (1-r)(1-2q) \Pr \{&\max(x_1, x_2) \geq a \mid \text{no shock,} \\ &\text{neither } R_1 \text{ nor } R_2 \text{ has power failure}\} \\ + 2(1-r)q \Pr \{&x_1 \geq a \mid \text{no shock, } R_2 \text{ has} \\ &\text{power failure but } R_1 \text{ does not}\}. \end{aligned}$$

This expression can be evaluated if estimates are available for p, q, r.

Discussion of Examples

In both of the examples just given, it is seen that the performance of a subsystem or component is not always summarized by the "probability of successful component performance." In this respect, the analysis of a whole system may frequently differ from the analysis of a subsystem. For instance, for the second hypothetical system, it is desirable to know in detail the conditional probability distribution of the output variable x_1 (on condition of no shock and no power failure). Indeed, "Pr {subsystem R_1 operates satisfactorily}" does not have a useful definition, since this probability depends on the performance of R_2.

It is suggested that analysis of subsystems is most usefully summarized by estimates of the probability distributions of its output variables under various conditions.

Confidence Interval for P

The preceding discussion has been directed toward the construction of a mathematical model of a system, incorporating information about interdependence among components. The outcome was an expression for the probability P of satisfactory system performance which could (in principle at least) be evaluated from estimates or conjectures for certain probabilities and conditional probability distributions. If the assumptions relating system performance to component performance were essentially correct, then the value of P could be predicted.

Suppose, however, that, as usual, only a limited number of components are available for making tests to estimate the required performance probabilities. Then, because of variability among components, the estimates are not very precise. It is desirable in such a situation to obtain also an interval estimate for P, so that the width of the interval can suggest how much uncertainty attaches to the estimated value of P. Such an interval estimate is called a *confidence interval.*[1]

A confidence interval for P is determined by a *confidence limit* P_0, smaller than the estimated value of P, which is derived by a method which insures that the statement "$P \geq P_0$" can be made at a specified *confidence level* (often 95 or 99 per cent). The confidence limit P_0 is determined by 1) the preassigned desired confidence level and 2) assumptions which have been made about the underlying probability distributions of the measurements which have been made to obtain an estimate of P. A fairly strict interpretation of a confidence interval is that if the data collection procedure could be repeated over and over, and if the same procedure were used each time to calculate (say) a 95 per cent confidence limit P_0; then in the long run the statement "$P \geq P_0$" made after each repetition of the procedure would be true 95 per cent of the time.

A looser interpretation of the statement, "$P \geq P_0$ at the 95 per cent confidence level," is that P_0 is the smallest value which is not extraordinarily unlikely in view of the measurements which have been made.

Illustration, for the System R

The usefulness of an interval estimate can be illustrated quite sharply in the context of the second illustrative example (the system R) of the preceding section. In order to estimate P for the system R, it is necessary to estimate three probabilities:

r = Probability of shock failure.
q = Probability of power failure in subsystem R_1 (or R_2) on condition no shock occurs.
p = Probability that one of the 100 components of R_1 (or R_2) works correctly on condition of no shock and no power failure.

Suppose, for simplicity, that only negligible uncertainty attaches to the estimates of r (based on a long history of weather data) and of q (based on long experience with the particular type of power source involved). Attention is focused on the problem of estimating p.

Tests have been made on 1000 components of the type to be used in the subsystems R_1 and R_2. None of them has failed to work correctly. Nevertheless, although this component is believed to be highly reliable, it is not believed to be perfect. Assuming that the "true" long-run proportion of satisfactory components of this type is p (less than unity), how small might p be without its being extraordinarily unlikely that every one of a sample of 1000 did not fail? The following table gives several answers to this question, corresponding to various confidence levels (*i.e.*, various meanings of "extraordinarily unlikely"). It also gives confidence limits for the probability p for sample sizes 500 and 5000, when no failures have occurred among the sample items.

CONFIDENCE LIMITS FOR p[9]

Confidence Level (In Per Cent)	Sample Size		
	500	1000	5000
95	0.9943	0.9970	0.9994
99	0.9908	0.9954	0.9991
99.5	0.9895	0.9947	0.9989

If the 99 per cent confidence level is adopted, then—since the sample size was 1000—the estimated confidence limit for p would be $p_0=0.9954$. The ordinary (point estimate) for the probability P for system R would be obtained using $p=1$; the only predicted failures would be shock and power failures. An interval estimate for P would be obtained by using p_0 instead of $p=1$; the result of this calculation would be a confidence limit P_0 for P.

The foregoing is admittedly a relatively simple example of a situation where a confidence limit for P could be calculated. In many cases, various approximations would be required, as well as difficult numerical integrations. The point is, that the calculation of a confidence limit for P is in principle possible once there exists a mathematical expression relating P to measurable aspects of component performance. And the calculation of a confidence interval provides reasonable protection against the imprecision of estimates based on relatively small samples.

Illustration, for the System S

Suppose that, for the system S of Fig. 1, the following confidence intervals have been obtained, each at the 99.5 per cent confidence level.

$$(1-r)\geq(1-r_0)\text{—probability of no shock,}$$
$$(1-q_{C_3})\geq(1-q_{C_30})\text{—probability } C_3 \text{ does not fail (on condition no shock),}$$
$$(1-q_D)\geq(1-q_{D0})\text{—probability } D_1 \text{ does not fail (on condition no shock).}$$

Suppose further that extensive experiments have verified that it is reasonable to assume that the (conditional) probability distributions of y_1, y_3, z_1, z_2 in the absence of catastrophic failure have means at "design center," known variances , and have the Gaussian form (but are truncated at upper and lower specification limits).

Now a method which suggests itself immediately for obtaining an approximate confidence limit P_0 for the system probability P, is to insert the confidence limits $(1-r_0)$, etc., in the expression for P. But each of the three confidence limits was obtained by a separate set of tests; each limit has confidence level 99.5 per cent, but a function of the three limits is subject to cumulative errors and must in general be less precise. Indeed, the confidence level for simultaneous assertion of the three confidence intervals is $(0.995)^3=0.990$, *i.e.*, 99 per cent. This is a lower bound for the confidence level for asserting "$P\geq P_0$."

If the functional form of P as a function of $(1-r)$, $(1-q_{C_3})$, and $(1-q_D)$ were taken into account, together with assumptions concerning the underlying probability distributions governing the measurements from which they are estimated, it would in principle be possible to improve on such a lower bound for the confidence level for asserting "$P\geq P_0$." One way of making such an improvement in a special case has been considered by Buehler.[10]

FURTHER USES OF A MATHEMATICAL EXPRESSION FOR P

In this section some additional uses are noted for a mathematical expression for the system probability P, constructed through the suggested approach.

Decisions About Design Changes

Given a mathematical expression for the probability P, incorporating information on interdependence of the components of a system, one may raise the following types of questions.

1) What would be the effect on P if a duplicate of a certain subsystem were added so that the system could operate if at least one of the subsystems did not fail?
2) Supposing that it has been decided to add redundant components, for which types of component should they be added?
3) What would be the effect of replacing a given component type by an improved type less frequently subject to catastrophic failure?

Many such questions can be answered usefully only in the context of costs: "Which is more costly, the present failure rate or the improvements required to achieve a lower failure rate?" The *effect* of certain kinds of changes of design may be calculated by a modification of the mathematical expression for P; the *costs* must be estimated from additional information. If several changes are proposed, each of which has presumably the same cost, they can be compared as to their effect on P. If it is specified that P must be improved by a fixed amount at least, manipulation of a mathematical expression for P can be helpful in deciding which design changes would be sufficient to accomplish this.

The approach to analysis of system performance suggested in this paper calls attention to the possibilities for isolating and determining the effect of particular types

[9] These values were computed directly since (in the case that no failures are observed in the sample) p_0 is obtained by solving for p in the equation $p^n=(1-\alpha)$, where n is sample size and 100α per cent is the confidence level. Charts giving confidence limits for the probability p for more general cases of the number of observed failures may be found in E. S. Pearson and H. O. Hartley, ed., "Biometrika Tables for Statisticians," vol. I, Cambridge University Press, Cambridge, England; 1954.

[10] R. J. Buehler, "Confidence Intervals for the Product of Two Binomial Parameters," unpublished paper; 1956.

of failure which may be removable, if their effect is great enough to justify the cost of making improvements.

Simulation of the Effect of Component Specifications

The attempt is generally made, in setting component specifications, to insure that every component which meets specifications will (in the absence of catastrophic failure) give satisfactory performance. But when the performance of one component is dependent on that of another in such a way that the first may or may not be satisfactory depending on the performance of the other, it may be difficult to determine realistic specifications.

Suppose, then, that a mathematical expression for the system probability P is available, relating system performance to the values of component characteristics, and taking account of interdependence among components. Then the same calculations which are carried out to obtain predictions of P on the basis of measurements made on components could be carried out using hypothetical "measurements" obtained by assuming that the probability distribution of a characteristic of a component has some assumed form (truncated at the specification limits).

Such calculations would make it possible to make predictions as to the effect of specification changes: in one case, the probability distributions of component characteristics might be unchanged, with only the truncation points (specification limits) altered; in another case, the whole probability distribution for a component characteristic might shift so as to have its mid-point at a new "design center"; in a third case, the truncation points might shift toward the center and the variance of a component characteristic decrease also, so that about the same proportion of components would still meet specifications.

The extreme case of these calculations—all component characteristics having their values at specification limits—is often unrealistic and can lead to either excessively stringent component specifications or excessively pessimistic predictions of system reliability. It is suggested that various "simulations" of the effect of component specifications can help to determine whether a given set of specifications is realistic.

Conclusion

The purpose of this paper has not been to outline detailed techniques for every step of the process of predicting reliability, but rather to suggest an approach to system analysis which would organize engineering design information and data on component performance in a way suitable for the application of probability theory and of the techniques of mathematical statistics.

If assumptions relating system performance to component performance and system design and use are essentially valid, one may obtain an estimated value for the system probability P and usually at least an approximate confidence interval reflecting the imprecision of this estimate.

In the long run, of course, the accuracy of a prediction based on an estimate of P can be determined only by experience with a system as a whole. It is believed, nevertheless, that the approach adopted in this paper may help to make possible some representations of system performance as a function of interdependent component performance which are sufficiently accurate to provide useful estimates of the system reliability.

Evaluation of Failure Data

HERBERT I. ZAGOR†

General

AMONG the many benefits for management which can be derived from a failure evaluation program are: 1) evaluating equipment and its maintenance procedures; 2) directing and guiding reliability improvement as well as future design; and 3) estimating spares.

Failure data obtained from one equipment may be used for reliability and spares predication for similar equipments under similar environments since operational comparisons may be made directly on such items. Increasing amounts of data are becoming available to allow correlation factors to be determined so that one can extend these predications to diverse equipments.

The problem of stocking spare parts resolves itself into setting up on a probability basis an equilibrium condition between the expected number of failures and the number of stocked items. Once spare parts estimates are available, a logistic system can be determined.

The main assumption is that the reliability of an equipment in its steady state behavior is such that those failures which inevitably occur are independent, random, and occur at a uniform rate. The failure dis-

† American Bosch Arma Corp., Garden City, N. Y.

tribution, therefore, is a Poisson distribution. In actual practice, the failures may not follow the Poisson distribution too closely. Instead, a two-parameter distribution such as the Polya or negative binomial may be more directly applicable. To obtain adequate data, a good failure-reporting system is required. Most benefit can be derived from such a system by having as small delay as possible in reporting of failures.

Theoretical Failure Patterns for Complex Equipment

Let us consider a pure-death process in which an initial population consisting of N_0 members is life tested, and no replacements are made for the failed elements. A plot for this situation of 1) the hazard, Z, and 2) number of failures per unit time $\Delta F/\Delta t$, as a

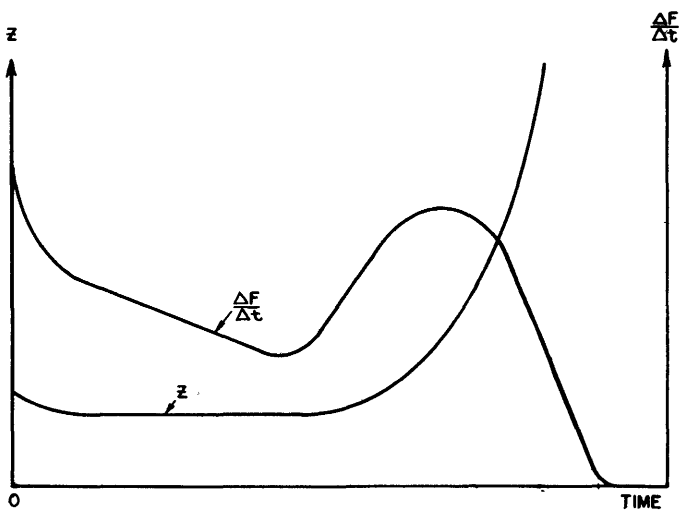


Fig. 1—Composite life pattern, pure death process.

function of operating time, is shown in Fig. 1. The hazard is equal to the ratio of number of failures per hour dF/dt, to number of survivors $S(t)$ at that time.[1] If for this statistical aggregate of N_0 elements the renewal of failed elements is taken into account, a birth-death process is obtained in which the failed elements are replaced as soon as they fail. A steady state condition eventually will arise in which the failure rate, and hence the replacement rate, becomes constant. The probability of a failure is a constant independent of the age of the population as a whole, and the exponential reliability law holds. This result holds if the elements are different or the same, or have chance or wear-out failures.[2] Two cases are to be considered, namely 1) the one-horse-shay in which at a time, τ_1 a large number of elements fail in a relatively short period of time due to excessive environmental and/or wear-out causes, and 2) no one-horse-shay at any time.

[1] R. R. Carhart, "A Survey of the Current Status of the Electronic Reliability Program," Rand Corp. RM-1131; August 14, 1953.
[2] *Ibid.*,

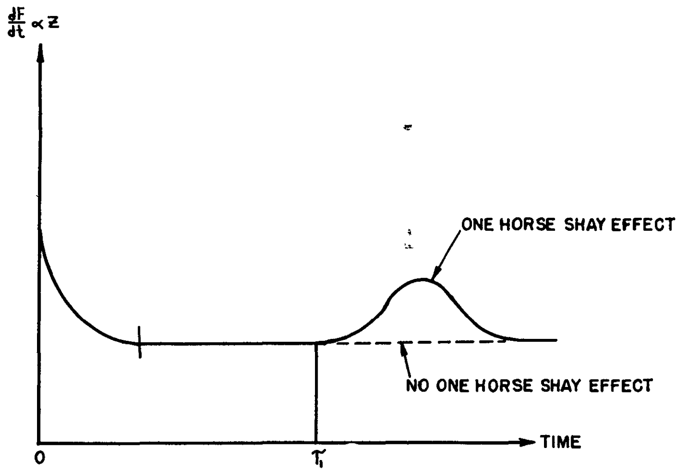


Fig. 2—Life pattern, renewal process.

The dotted line in Fig. 2 shows a failure pattern iu which the hazard Z is plotted as a function of time for a renewal situation without the one-horse-shay effect The solid curve in Fig. 2 shows the one-horse-shay effect in this failure pattern, which is also illustrative of a renewal situation in which preventive maintenance techniques are employed at time τ_1.

It should be noted that dF/dt is proportional to the hazard, Z, as well as of Y, the fraction of initial sets failing per hour, since $N = N_0 =$ a constant.

Equipment, both military and industrial, generally falls into the category of a renewal process except for throw-away items so that Fig. 2 may be taken as the model for a complex equipment aggregate such as a computing laboratory installation.

Two Statistical Functions Useful in Failure Predication

Poisson

The Poisson distribution arises in describing the frequency of random, independent events whose probability of occurrence in any given interval of time is constant. The probability of observing n events is given by the expression

$$P(n) = e^{-m}m^n/n! \tag{1}$$

where m is the expected or average number of occurrence of events.

It is a function of one parameter, the mean m, or $P = f(m)$. All that is needed is m, the average number of events which occurred in the past, in order to compute the probabilities of the occurrence in the future of various numbers of these events. The mean and variance of the Poisson distribution are equal so that curve fitting can be accomplished by fitting to the mean, which is a constant.

The exponential law is a special case of the Poisson relationship since $P(0) = e^{-m}$, which is the probability of zero failure during the unit time interval. Hence reliability may be defined in one way as $P(0)$ or the probability of zero failure over the unit time interval, having observed m failures, on the average, previously.

Polya or Negative Binomial

The Poisson distribution is based on the constancy of the expected number, m, from trial to trial. If m, varies, but the mechanism causing the events remains basically Poisson, then a modification such as the negative binomial is required to describe this situation. The negative binomial distribution may be derived as follows.[3]

Suppose a distribution of chance follows the Poisson law but that m itself is unknown, having a distribution of chance given by the Pearson Type III law

$$d\alpha = \frac{c^p\, m^{p-1}}{\Gamma(p)} e^{-cm} dm. \tag{2}$$

To get the total probability for any value of n we must integrate for all possible values of m. The generating function is

$$\pi(\psi) = (c/c+1)^p\,(1-\psi/(c+1))^{-p} \tag{3}$$

the coefficients of ψ^n being the frequency of 0, 1, 2, $\cdots$, successes, *viz.*,

$$(c/c+1)^p(1,\ p/(c+1),\ p(p+1)/2!(c+1)^2,$$
$$p(p+1)(p+2)/3!(c+1)^3 \text{ etc.}) \tag{4}$$

which are the successive coefficients of ψ^n. The mean, $\overline{X}$, is given by p/c and the variance, σ^2, is $p(1+1/c)/c$. It can be seen that (3) is a negative binomial function with multiplying factor $(c/c+1)^p$.

It is of interest to note that the negative binomial distribution also arises from the Polya Urn problem.[4] In the Polya Urn problem, an urn contains b black and r red balls, and random drawings are made. The ball drawn is always replaced, and, in addition, c balls of the color drawn are added to the urn. This scheme was devised for analysis of phenomena like contagious diseases, where the occurrence of certain events increases their future probabilities.

If the first ball is black, the conditional probability of a black ball at the second drawing is $(b+c)/(b+c+r)$. The absolute probability of the sequence black, black, is therefore $(b/(b+r))(b+c/(b+c+r))$ from the relationship for the joint probability

$$P(AH) = P(A \mid H)\cdot P(H). \tag{5}$$

If the first two drawings result in black, then the urn contains $b+r+2c$ balls among which $b+2c$ are black. The conditional probability of a black ball at the third trial becomes by (5), $(b+2c)/(b+2c+r)$. In this way all probabilities can be calculated. Any sequence of n drawings resulting in n_1 black and n_2 red balls $(n_1+n_2=n)$ has the same probability as the event of extracting first n_1 black and then n_2 red balls, namely

$$p_{n,m} = \frac{b(b+c)(b+2c)\cdots(b+n_1c-c)r(r+c)\cdots(r+n_2c-c)}{(b+r)(b+r+c)(b+r+2c)\cdots(b+r+nc-c)}. \tag{6}$$

The Polya Urn scheme, in the limit, goes to the Polya process. Let drawings be made at the rate of one in time h and let $h\to 0$, $n\to\infty$ so that $np\to t$, $nv\to at$, where $p=r(r+b)$ and $v=c(r+b)$. Then making this limit passage in (6), we obtain for $n\geq 1$ and $n=0$ respectively,

$$\pi_n(t) = \left[\frac{t}{1+\frac{t}{p}}\right]^n \cdot \frac{1\left(1+\frac{1}{p}\right)\cdots\left(1+[n-1]\frac{1}{p}\right)}{n!}\,\pi_0(t) \tag{7}$$

$$\pi_n(t) = \left(1+\frac{t}{p}\right)^{-p}\left[1-\frac{at}{1+at}\right]^{-1/a} \tag{8}$$

and

$$\pi_0(t) = \left(1+\frac{t}{p}\right)^{-p}. \tag{9}$$

If $t=p/c$, $a=1/p$, then (8) becomes

$$\pi_n\left(\frac{p}{c}\right) = \left(\frac{c}{c+1}\right)^p\left(1-\frac{\psi}{c+1}\right)^{-p}, \tag{10}$$

which is identical with (3).

In addition, the Polya distribution may be defined by considering directly a random process with continuous time parameter, a process which is characterized by an intensity function

$$\lambda_n(t) = (1+an)/(1+at). \tag{11}$$

The Polya distribution then is the solution to the differential equations

$$P_n'(t) = -\lambda_n P_n(t) + \lambda_{n-1}P_{n-1}$$
$$P_0'(t) = -\lambda_0 P_0(t) \tag{12}$$

with initial condition $P_0(0)=1$, and λ_n given by (11).

[3] M. G. Kendall, "The Advanced Theory of Statistics," Hafner Publishing Co. New York, N. Y., pp. 124; 1952.

[4] W. Feller, "An Introduction to Probability Theory and Its Applications," John Wiley and Sons, New York, N. Y.; 1952.

We obtain for this solution

$$\pi_n(t) = t^n(1 + at)^{-n-1/a} \cdot \frac{(1 + a)(1 + 2a) \cdots [1 + (n - 1)a]}{n!}$$

$$\pi_0(t) = (1 + at)^{-1/a}. \qquad (13)$$

If we make the transformations $t = p/c$ and $a = 1/p$, then (13) becomes identical to (3). Thus the Polya process is a nonstationary pure birth process with λ_n given by (11). The negative binomial is a function of two parameters, the mean $\overline{X} = p/c$ and the variance, $\sigma^2 = p/c(1 + 1/c)$. Curve fitting is done by matching the observed data to the mean and variance.

Determination of Poisson Distribution Range Limits

If the occurrence of failures are assumed to obey a predetermined statistical distribution, then it should be possible to identify those elements which have failure probabilities significantly different from those of the average element.

Identification of these abnormal elements will point out those areas in which one should check circuit application and maintenance techniques, with the abnormal items being those which require first attention. As a result it should be possible to monitor equipments for reliability, and establish a reliability improvement procedure.

For the Poisson distribution the functional relationship between the limits of the range, c, and the mean number of failures, m, of the average element can be plotted as a graph by use of Molina's Poisson tables.[5] These tables contain for a given mean, m, the total cumulative probability of there being c_1 or more failures, as well as the total cumulative probability of there being c_2 or fewer failures. Two probability values such a 1/1000 and 1/100 are generally employed, and define four curves, namely, the two upper curves Action (A) and Warning (W) at the 1/1000 and 1/100 probability levels and the two lower curves warning (w) and action (α) at the 1/100 and 1/1000 probability levels respectively. These curves are shown in Fig. 3.

Example

Comparison of Performance of Electron Tubes in Three Makes of Analog Computers:[6] In the case of complex electronic equipment it has been shown experimentally that electron-tube failures usually follow a Poisson distribution. Since failures occur at random, a variation in the number of failures in each tube type in the same period will be expected. The range for determination of abnormality is shown in Fig. 3.

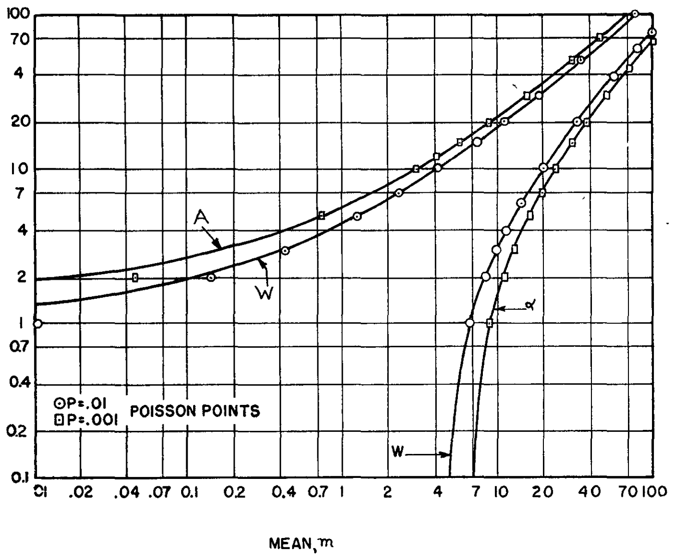


Fig. 3—Poisson cumulative curves of range limits, C, vs mean, m. The four curves represent the probability limits $A = 1/1000$, $W = 1/100$, $w = -1/100$, and $\alpha = -1/1000$.

If equal numbers of tube types are used in the equipment, then the expected mean and range limits will be the same for each of these tube types. However, in an actual case the number of sockets will vary with the tube types, so that the expected means will be proportional to the number of sockets, with the range limits varying accordingly.

In Table I (next page) there are 1835 sockets with a total of 1361 failures or a first mean rate of 1361/1835 = 0.742 failure/socket. The expected mean for each tube type is calculated by multiplying the mean rate by the number of sockets for the tube type. For each tube type, the A, W, w, and α range limits are taken from the graph in Fig. 3. It can be seen that the 6SN7 and 6L6 in B are out of limits and so are considered as abnormal. The 6SJ7 in A, and 6SL7 in the B are high, and so should be suspect as a result of the first mean rate calculation.

If the data include failures from abnormal tube types, as well as from the average types, the value of the mean calculated from all the data can be shifted by the failures of the abnormal types. One extremely high failing type can mask a few moderately high failing types and even force a few average types that happen by chance to be a little low to appear abnormally low. The adjusted figures are calculated by leaving out tube types 6SN7 and 6L6 in the B which tested as very high on the first calculation. The adjusted data show that the 6SJ7 in the A and the 6SL7 in the B are now abnormal, and all the remaining tube types are within limits.

[5] E. C. Molina, "Poisson's Exponential Binomial Limit," D. Van Nostrand Co., New York, N. Y., 1942.

[6] F. A. Hadden, "Machine Testing for Deviation of Data from a Poisson Distribution," *Communication and Electronics Magazine*, p. 155; May, 1955. This paper discusses the use of a machine in solving a problem of this type.

TABLE I

RANGE LIMITS FOR ELECTRON-TUBE FAILURES FROM THREE TYPES OF COMPUTING AMPLIFIERS

Amplifier Chassis Type	Tube Type	Observed Number of Failures	Sockets per Chassis	Total	Expected Mean	W	w	A	α	Adjusted Expected Mean	Adj. W	Adj. w	Adj. A	Adj. α	Test Result
A	6SJ7	286	6	408	303	345	263			231			278[1]		*A*
	6AC7	31	2	136	101				71	77				51	OK
	5691	12	2	136	101				71	77				51	OK
	5881	75	2	136	101		78		71	77	98	57			OK
B	6SN7	259	2	194	144			183[1]							*A*
	6SL7	284	4	388	288	329	248			220			248[1]		*A*
	6L6	283	2	194	144			183[1]							*A*
C	6SJ7	131	3	243	180				140	137	165	110			OK
Total		1361		1835											

First Mean Rate $=\frac{1361}{1835}=0.742$ Failure/socket.

Adjusted Mean Rate $=\frac{1361-259-283}{1835-194-194}=\frac{819}{1447}=0.566$ failure/socket.

[1] Tested out of range, *i.e.*, observed number $>A$.

DEFINITIONS OF PROBLEM OF SPARES DETERMINATION

The integral

$$\int_0^T \frac{dF}{dt}\,dt,$$

or the area under the curve of number of failures per unit time as a function of operating time, is F, the total number of failures. What is desired in an over-all logistics scheme is an equilibrium condition between the replacement or stocking up of spares and the total number of failures or demand.

To do this, a statistical function or distribution describing the failure or demand pattern is determined, whence this function is used as a means for estimating future demand so that, on a probability basis, an equilibrium situation can be set up.

METHODS OF SPARES DETERMINATION

Cumulative Probabilities

The reliability, or probability of zero failure per unit of time, assuming a Poisson distribution, is given by $P(0)=e^{-m}$. Then knowing from past data the average failure rate m, an estimate for spare parts can be made from the unreliability of such an aggregate. The unreliability is

$$u = 1 - P(0) = (1 - e^{-m}) \tag{14}$$

and represents the cumulative probability of 1, 2, 3, $\cdots$, failures occurring.

For N elements and an operating or use time of t hours, the expected number of failures would be given by

$$UNt = N(1 - e^{-mt}). \tag{15}$$

Therefore, UNt elements would be the expected number of elements which need replacement after each t hours of use. These UNt expected failures can be the basis for a spare parts program, as well as maintenance program.

The reliability R may also be written as

$$R = 1 - \sum_{i=1}^{n} P(i).$$

The unreliability, or $\sum_{i=1}^{n} P(i)$, is the cumulative probability of 1, 2, 3, $\cdots$, n failures occurring. For N elements, the unreliability is $N\sum_{i=1}^{n} P(i)$. This is, however, precisely the number of expected failures, which is of course, equal to the spares required. For any average value, m, there is a unique Poisson distribution. From Molina's Table II of Cumulated Probabilities,[5] we have $\sum_{i=1}^{n} P(i)$ computed which, when multiplied by N, gives the number of spares required.

In an actual problem, failure data for a component or group of equipments will be tabulated as N failures over a time period of T units for n similar components or equipments. Thus for a sample size n, the expected number of failures, a, over time period, T, is given by $a=N/T$ This is equivalent to the previous calculation, since $a=np=m$, where p is the failure probability per component or equipment per unit of time, n is the sample size, and a is Molina's a, or expected number of failures in a sample of n elements. However, another factor must now be taken into account. For each value of m, which is the expected number of failures or observed demand rate, there will be associated a range of demands with a range of probabilities. Fig. 3 shows this range as a function of m. Hence for any given demand m, the limit values m_2 and m_3 at probability values such as 10^{-2} or 10^{-3} can be determined so that a stock supply of quantity m_2 will not be expected by a demand 99 times out of 100, or quantity m_3 999 times out of 1000.

Thus, even though the actual demand or failure rate m_1 has been observed, nevertheless a stockroom should

TABLE II

Table for Limits $L\alpha_{1,2}$ of Confidence Intervals for Poisson Parameters, m

	LOWER ($L_{\alpha_1}(m)$)												UPPER ($L_{\alpha_2}(m)$)											
m	0.005	0.025	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10	0.10	0.09	0.08	0.07	0.06	0.05	0.04	0.03	0.02	0.01	0.025	0.005
0	0	0	0	0	0	0	0	0	0	0	0	0	2.303	2.483	2.527	2.660	2.814	2.996	3.220	3.569	3.913	4.654	3.69	5.30
1	0.005	0.0253	0.0105	0.0202	0.0305	0.0408	0.0513	0.0619	0.0726	0.0834	0.0943	0.1054	3.890	4.022	4.169	4.334	4.523	4.745	5.013	5.357	5.835	6.640	5.57	7.43
2	0.103	0.242	0.1485	0.2147	0.2675	0.4124	0.3531	0.3937	0.4295	0.4644	0.4993	0.5309	5.323	5.475	5.643	5.831	6.046	6.296	6.599	6.984	7.517	8.406	7.22	9.27
3	0.338	0.619	0.4321	0.5643	0.6624	0.7441	0.8167	0.8815	0.9409	0.9983	1.0503	1.1020	6.681	6.849	7.035	7.242	7.479	7.754	8.086	8.505	9.085	10.047	8.77	10.98
4	0.672	1.09	0.8210	1.0149	1.1530	1.2668	1.3650	1.4527	1.5331	1.6083	1.6778	1.7439	7.994	8.176	8.377	8.601	8.857	9.154	9.511	9.962	10.581	11.648	10.24	12.59
5	1.08	1.62	1.2773	1.5279	1.7057	1.8468	1.9662	2.0776	2.1759	2.2667	1.3516	2.2334	9.275	9.470	9.685	9.925	10.197	10.513	10.893	11.371	12.028	13.109	11.67	14.15
6	1.54	2.20	1.7841	2.0884	2.3004	2.4682	2.6125	2.7390	2.8539	2.9598	3.0584	3.1512	10.533	10.739	10.967	11.221	11.508	11.843	12.243	12.747	13.437	14.571	13.06	15.66
7	2.04	2.81	2.3286	2.6833	2.9268	3.1206	3.2848	3.4289	3.5590	3.6788	3.7902	3.8946	11.771	11.989	12.229	12.495	12.798	13.149	13.568	14.095	14.817	16.000	14.42	17.13
8	2.57	3.45	2.9569	3.3068	3.5805	3.7978	3.9802	4.1405	4.2854	4.4177	4.5403	4.6555	12.995	13.223	13.474	13.753	14.069	14.435	14.874	15.048	16.212	17.469	15.76	18.58
9	3.13	4.12	3.5070	3.9518	4.2549	4.4942	4.6951	4.8704	5.0283	5.1728	5.3070	5.4320	14.206	14.444	14.705	15.996	15.371	15.747	16.192	16.774	17.570	18.831	17.08	20.00
10	3.72	4.80	4.1290	4.6176	4.9475	5.2074	5.4248	5.6153	5.7860	5.9458	6.0861	6.2209	15.450	15.693	15.936	16.262	16.615	16.969	17.486	18.030	18.861	20.178	18.39	21.40

stock m_α of these items, where $m_\alpha > m_1$, so that demand will not exceed supply within a probability of p_α. We can equally well choose 10^{-2}, 10^{-3}, or any other probability value, depending upon the over-all system factors and the consumer risk that has been chosen. For example, a probability level of 10^{-3} requires a stock of m_3 items. Incidentally, the observed failure or demand rate m_1, if stocked in a stockroom, would only give assurance that on the average, the probability of demand exceeding spares would be 0.53. This is a consumer risk of 53 per cent.

Confidence Intervals

It should be noted that the Poisson parameter, m, is at best subject to sampling error, since the observed value of demand is the result of a number of sample observations. Each demand, or failure group, per unit time may be considered as a new sample. Hence, the true value of the Poisson parameter, m, can only be stated within a given confidence level. Table II of confidence intervals for Poisson parameters, m, are taken from Youngs.[7]

Conditional Probabilities

The conditional probability, $P(y|x)$ is defined as the probability of observing the occurrence of event y after event x has been observed, or the relative frequency y occurs knowing that x has occurred. The basic relationship in conditional probability is given by (5): the conditional probability of event y on the hypothesis x is given by the ratio of the joint probability of (y, x), to the probability of hypothesis x. The Poisson distribution gives the probability of observing 0, 1, 2, 3, $\cdots$, events, having observed m events, on the average, previously. Thus each of the terms P_i in a Poisson distribution is a conditional probability and we may write $P_i = P(y_i|m)$.

The conditional probability function for the negative binomial distribution (3) is given by the expression[8]

$$\pi(y|x) = \left(\frac{c+1}{c+2}\right)^{p+x}\left(1 - \frac{\psi}{c+2}\right)^{-(p+x)}. \tag{16}$$

Successive terms π_{ij} in (16) give the conditional probability of observing y_i events when x_j events have been observed in the past, assuming the Polya distribution as the basic distribution of the x_j events.

Discussion

The inventory in a stockroom consists of many items. If we assume a one-to-one correspondence between number of failures, *i.e.*, types of items that failed and items drawn from inventory stock, we can use stockroom inventory as our measure for spares provisioning. Since the demand for individual items may be low, *i.e.*, 0 or 1, over long periods of time, the negative binomial distribution should be expected also to describe the demands of individual items more closely than the Poisson. This is so since a statistical description of a series of events which are nearly Poisson distributed, but which have long tails, *i.e.*, low frequency requirements over long periods of time, may be approximated by the negative binomial distribution.

Example

Let us consider the problem of estimating the number of spare amplifiers required in a computing laboratory which consists of three different makes of analog computers.[9]

[7] J. W. T. Youngs, M. A. Geisler, and A. R. Mirkovitch, "Confidence Intervals for Poisson Parameters in Logistics Research," Rand Corp. RM-1357; September 30, 1954.

[8] J. W. T. Youngs, M. A. Geisler, and B. B. Brown, "The Prediction of Demand for Aircraft Spare Parts Using the Method of Conditional Probabilities," Rand Corp. RM-1413, January 17, 1955.

[9] Problem taken from work done by H. I. Zagor while consultant at Convair, San Diego, Calif.

Analysis of Problem: The amplifiers have been chosen for analysis to illustrate the methods of estimating spares. Failure records had been kept of the computers since their installation in the computing laboratory early in 1954. The failure data were on cards which recorded, for each amplifier chassis, the date 1) on which the amplifiers had failed, and 2) on which the amplifier was returned to service, plus the diagnosis and cure required to make it operational.

In any interval of time Δt, let

A_f = number of amplifiers that fail.
A_n = number of amplifiers that are being repaired.
A_s = number of amplifiers that are in storage.
A_{t_0} = state of amplifier failure—replacement—repair storage system cycle at time t_0.
A_{t_1} = same at time t_1, where $t_1 > t_0$.

Then $A_f + A_n + A_s$ = a constant, and $A_{t_0} = A_{t_1}$. Hence the amplifier—replacement—repair—storage system cycle is a stationary process in which the total number of amplifiers under consideration remain a constant. The failure—replacement portion of the system is a renewal process since amplifiers, when they fail, are replaced by good amplifiers from storage.

The assumption is that, on the average, each amplifier of the total amplifier population is used operationally for approximately an equal length of time.

The cards containing the failure records were reviewed critically and only those cards selected which were complete enough to have meaningful data.

Type A Amplifiers

Cards for 68 amplifiers out of a total population of 75 amplifiers were found to have sufficient data for analysis. These cards were divided into four sample groups, the first three groups consisting of 20 amplifiers each, and the fourth group containing the residue of eight amplifiers. Plots of number of failures per four week interval vs calendar time were made for each of the groups and are shown in Fig. 4(a), 4(b), and 4(c). It can be seen that each of the three amplifier groups yielded substantially an identical plot. Group four contained too few failures itself for statistical results, and so was lumped into the group three data. In order to get meaningful statistical data, a four-week time base interval was chosen.

Fig. 4(c) shows the cumulative total. The initial failure peak is clearly visible, followed by a valley, and then a plateau. Thus the general shape of the curve is similar to Fig. 2.

It is very probable that the plateau region may still be part of the valley region, as possibly more stringent operating criteria may have been applied at that later time so that what was acceptable performance previously is no longer acceptable. This is to be expected as personnel become better trained and more familiar with the equipment. The peak in the 20th interval was reduced as indicated by subtracting the obviously bad

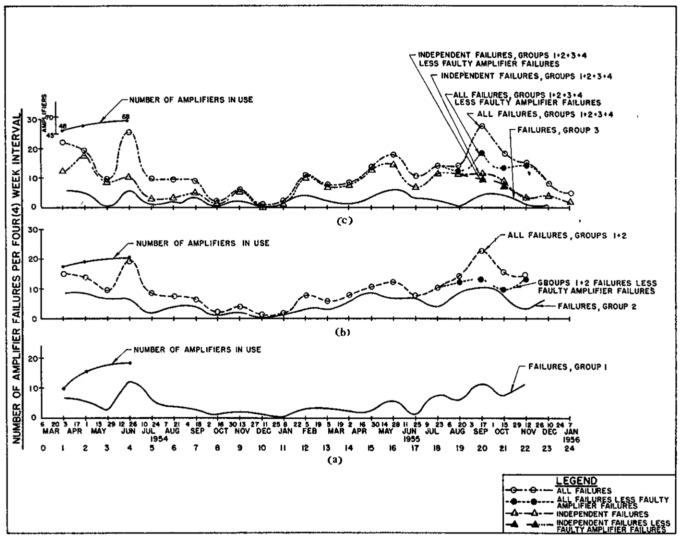


Fig. 4—Plots of number of amplifier failures per four-week interval vs time for type *A* amplifiers. (a) Total failures, group 1 amplifiers. (b) Total failures, group 2 amplifiers; sum of groups 1+2 amplifiers. (c) Total failures, group 3+4 amplifiers; Sum of groups 1+2+3+4 amplifiers.

amplifiers. Reliability, of course, can only be assessed against those items which have a chance of success.

After the tabulation of total failures vs time, the data are separated into categories such as 1) independent failures, 2) apparent failures which subsequently showed acceptable operation on bench check, and 3) initial failures. In addition, component failures such as tubes by type, choppers, are tabulated.

An independent failure is classed as one which happens of its own accord, but not as a result of another failure. Since the failure records were not complete, judgment was required. The following criteria were applied in determining independence of failure.

1) No external causes apparent, such as defective power supply, mistreatment.
2) Not a mistake on the part of a technician, in that an apparently faulty amplifier is removed and subsequently found to check out good on the bench.
3) No initial failure, such as wrong tube inserted, a lead not connected, poor solder joint, or mechanical failure.
4) Consistent failure of a power tube, such as type 5687, would represent a failure due to mistreatment, or design fault.
5) Defective voltage tubes such as types 6SJ7, 6AC7, 5691 were considered independent failures, except when otherwise indicated.
6) Those cases in which an amplifier had one or more component failures were still classified as one amplifier failure, and independent, unless the cause of failure fell into one of the other categories. There is a finite probability, of course, that one or more failures can occur during a given interval of time in a given amplifier.

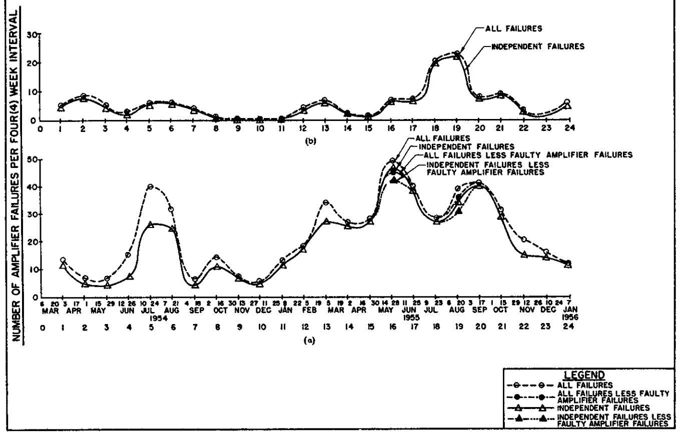


Fig. 5—Plots of number of amplifier failures per four-week interval vs time. (a) Type *B* amplifiers. (b) Type *C* amplifiers

Types B and C Amplifiers

Similar graphical plots of number of failures per four week interval vs time are shown in Fig. 5(a) and (b) for the *B* and *C* amplifiers. The *B* and *A* amplifiers can be compared directly since each consists of modules which contain two amplifiers on one chassis, and was installed at about the same time and used for approximately the same length of time.

The curves for the *C* amplifiers, in Fig. 5(b), are quite different from those in Figs. 4(c) and 5(a). The *C* amplifiers had been in use approximately for six to nine months prior to installation of the *A* and *B* amplifiers and had undergone extensive marginal testing and preventive maintenance by March, 1954, when the other amplifiers were installed. Hence, by March, 1954, the conditions for a stationary process should have been met, in the case of the *C* amplifiers.

Examination of Fig. 5(b), shows no initial peak, but rather an approximate constant level for amplifier failures until the peak in the 18–19 periods. After the peak, the failure rate settled down once again to approximately the same level as previously observed. This is similar to that portion of Fig. 2 which follows the initial failure period.

The peaks which appear in all the curves at approximately the 20th period are an illustration of the one-horse-shay effect in that additional failures were probably caused as a result of the unusual heat wave in San Diego during the summer of 1955, when the equipment ran hotter than allowable tolerances.

Techniques of Solution

Determination of Distribution Function on Amplifiers: Independent failures for each of the 68 chassis for nine 8-week periods over the interval June 26, 1954 to December 9, 1955 were recorded. The 612 chassis-periods were divided into five groups, namely, those with 0, 1, 2, 3, and 4 failures respectively. This observed distribution was then fitted to Poisson and negative binomial distributions respectively. The results are shown in Table III.

TABLE III

Results of Fitting Failure Data of *A* Amplifiers to Poisson and Negative Binomial Distributions

Number of Independent Failures per Chassis-Period	Number of Chassis-Periods Having Stated Number of Failures		
	Observed	Poisson	Negative Binomial
0	499	492	500
1	97	107	94.2
2	13	11.6	15.2
3	2	0.838	2.30
4	1	0.0454	0.336
Total	612	611.4834	612.036

The total number of independent failures is 133. Hence average number of failures per chassis-period is $\overline{X}=133/612=0.217$. This value of 0.217 is set equal to m in (2), and P_i computed. For the negative binomial frequency distribution, the mean of the observed data is $\overline{X}=0.217$, and the variance from the mean of observed data is $\sigma^2=0.250$. For curve fitting, the first and second moments are matched to the observed data. Thus, $\overline{X}=0.217$, and $\sigma^2=0.250$. Solving these simultaneous equations, we obtain $p=1.43$, and $c=6.57$.

For the negative binomial distribution, (3), the probabilities are $\pi_0=0.817$, $\pi_1=0.154$, $\pi_2=0.0243$, $\pi_3=0.00376$, $\pi_4=0.00055$ etc. Multiplication of each term by 612 gives the expected frequency.

χ^2 *Test:* Since the χ^2 test cannot be safely applied when the expected frequency in any cell is less than five, sufficient cells are lumped together so as to get a frequency total of five or more. In the Poisson case, lumping of the last three frequencies resulted in $11.6+0.838+0.0454=12.5$. The value for χ^2 becomes 2.07. For the negative binomial, the value for χ^2 becomes 0.267. In the Poisson case, the mean is determined from the observed data, and the Poisson distribution fitted accordingly. Hence, the number of degrees of freedom is given by the total number of cells minus two.

The negative binomial distribution is a two-parameter distribution. In the illustrative problem both the mean and variance were determined from the observed data. Hence, for this case the number of degrees of freedom is given by the total number of cells minus three.

In the illustrative problem, after pooling the last three cells together, only three cells remain, which leaves zero degree of freedom for the negative binomial distribution. Hence a χ^2 test cannot be made on these data. Nevertheless, inspection of the observed and expected frequencies shows a closer fit with the negative binomial than with the Poisson. The probability value for goodness of fit for the Poisson case, $\chi^2=2.07$ and two degrees of freedom, is 0.3 or 30 per cent. This cannot be done for the negative binomial fit to the observed data.

Remarks

It may be seen that the negative binomial distribution describes very well the observed distribution of independent failures on 68 A chassis for nine eight-week periods, June 26 to December 9, 1955 inclusive. In a practical case, such as the example shown here, the agreement between the observed frequencies and the Poisson distribution is very adequate, indeed, for any computation, and no additional attempt at curve fitting to another distribution would be necessary. However, for the purpose of an example and to illustrate the methods, we have gone ahead and fitted the negative binomial. All future calculations will be made for both these distributions to illustrate various methods of estimating spares.

In the illustrative problem, as indeed in problems in general, one's good judgment should prevail as to what constitutes an adequate solution, assuming one exists. Thus any results obtained from the mathematics should be tempered by sober judgment before one plunges ahead. In other words, *caveat emptor* should be observed in mathematics, as well as in economics.

Method of Cumulative Probability

In the illustrative problem, assume data are adequately defined by a Poisson distribution in which $m=133$ failures/68 amplifiers/72 weeks or 0.0272 failure/amplifier/week.

The reliability or probability of one amplifier having zero failure in one week, having observed in the past 0.0272 failure per amplifier per week, is given by $P_0=e^{-0.0272}=0.973$. The unreliability or probability of observing one or more failure in one amplifier during one week is given by $U=1(1-e^{-0.0272})=0.027$.

The expected number of amplifier failures per week for 68 amplifiers is given by $UN=0.027\times 68=1.836$ amplifier. Hence, to achieve an equilibrium situation with respect to amplifiers, we would merely have to supply a stockroom with 1.836 spare amplifier per week in order to have an expectancy of zero downtime for equipment containing 68 amplifiers. In actual practice, the values obtained for the spares would be rounded off to the next higher integer, or two amplifiers in our case.

An alternative method for computing cumulative probabilities is as follows. The average failure rate per amplifier per week is given by $m=0.0272$. For this value of m, or a in Molina's tables, we can compute $\sum_{i=1}^{n} P(i)$ or the cumulative probability of observing 1, 2, 3, $\cdots$, failures, having previously observed m failures on the average. From Molina's tables, this is equal to 0.027. For 68 amplifiers, the number of expected failures/week is $68\times 0.027=1.836$.

It can be seen that the probability calculations yield a more economical result than the simple average failure rate, namely 1.836 vs 1.85. This difference is more pronounced for higher failure rates. For example, assume a failure rate ten times as large as the observed value, or $m=0.272$ failure/week/amplifier. The probability calculations would yield $68\times 0.238=16.2$ spares required as compared with 18.5 spares required on a simple average basis.

Consumer Risk Considerations

The value for the spares, *i.e.*, 1.836, which we have previously obtained by cumulative probability methods is that number of spares which has a probability of 0.53 of being exceeded by failures, or represents a consumer risk of 0.53.

From Fig. 3, $m=1.85$ and consumer risk levels of 10^{-1}, 10^{-2} or 10^{-3} imply 5, 7, or 8 spares required per 68 amplifiers per week, respectively.

Method of Confidence Intervals

In the illustrative problem, $m=1.85$ failure/68 amplifiers/week. For 90 per cent confidence limits, the content in Table II is searched for values within any limits such that $1-(L_{\alpha_1}+L_{\alpha_2})=0.9$.

For symmetrical limits, $L_1+L_2=0.05$. From the tables we obtain the lower limit $L_1(0.05)=0.32$, and the upper limit $L_2(0.05)=5.8$.

Since in stocking spares we are concerned with demand not exceeding inventory, the upper value is taken. Hence, within a 90 per cent confidence limit, for symmetrical limits, we should stock 6 spares per week for the 68 amplifiers if we have observed previously a demand of 1.85 amplifier per week per 68 amplifiers.

It should be noted that any confidence limits desired may be chosen and the spares computed similarly.

Method of Conditional Probability

When the observed events can be described satisfactorily by a Poisson distribution, any of the methods discussed can be used, depending upon the specific situation and the nature of the problem. Since the Poisson distribution is basically a conditional probability function, the cumulative probability methods discussed have been based upon conditional probabilities.

However, if the data are described more adequately by a negative binomial distribution, the conditional probability function for the negative binomial distribution is required. Thereafter, cumulative probability techniques can be applied to estimate the spares requirements in a manner similar to that illustrated. In the illustrative problem, $p=1.43$ and $c=6.57$, hence, the conditional probabilities of observing 0, 1, 2, 3, etc. events having observed 1.85 failure/68 amplifiers per week are $\pi(0|1.85)=0.66$, $\pi(1|1.85)=0.252$, $\pi(2|1.85)=0.062$, $\pi(3|1.85)=0.013$, $\pi(4|1.85)=0.00185$, $\pi(5|1.85)=0.00031$, etc. From these computations, a cumulative probability of approximately 10^{-3} that demand should not exceed spares requires a stock of about five amplifiers.

TABLE IV

Required Number of Spares for A Amplifiers

Expected Number of Demand/68 Amp/Week	Poisson	Negative Binomial
Consumer Risk		
0.53	2	2
10^{-1}	5	2
10^{-2}	7	3
10^{-3}	8	5
Confidence Limits		
90 per cent	6	

TABLE V

Sample Sizes, and χ^2 Test Results for Fitting of Failure Data to Poisson Distribution for A, B, C Amplifiers

Chassis Type	No. of Amplifiers in Sample	Total No. of Amplifiers	Expected No. of Failures per 8 Week Period	χ^2 Acceptance Level Poisson
A	68	75 dual	14.8	0.3
B	102	105 dual	44.5	approx. 0.015
C	82	89 single	11.3	0.3

TABLE VI

Required Number of Spares for A, B, C Amplifiers as a Function of Consumer Risk Level

Chassis Type	Sample Size	No. of Independent Failures in 72-Week Period	Molina's a, Expected No. of Failures, Normalized to 68 Dual Channel Amplifiers per Time Unit		Molina's c, Number of Spares Required					
			Day	Week	Daily Basis Consumer Risk			Weekly Basis Consumer Risk		
					0.1	0.010	0.001	0.10	0.01	0.001
A	68	133	0.308	1.85	2	3	4	5	7	8
B	102	401	0.618	3.81	3	4	5	7	10	12
C	82	102	$0.197k$	$1.18k$						
		For $k=2$,	0.394	2.36	2	3	4	5	7	9
		For $k=2.6$,	0.513	3.06	2	4	5	6	9	11

Results

The results of the methods for A amplifiers are shown in Table IV. It may be seen that the results of these various calculations are not identical, even though approximately the same. This means that in general, one should evaluate the over-all physical situation for the problem at hand before accepting any one method as final. In the example cited in the text there were not a sufficient number of independent cells to fit a negative binomial. Hence, since the Poisson gave an adequate fit, any of the methods employed with the Poisson function would be adequate.

Analysis of Data Obtained on Independent Failures for B and C Amplifiers, and Determination of Spares.

The data obtained on independent failures for the B and C amplifiers were analyzed to see if they followed a true Poisson or negative binomial distribution. The failure data were on a time unit of 8 weeks in order to have statistically meaningful numbers. The calculated distributions for the Poisson and negative binomial were compared with the observed distribution, and the χ^2 test made, where applicable. The results are shown in Table V.

If the distribution of failures is assumed adequately described by the Poisson function, spares can be calculated by the method of cumulated conditional probability. In Table VI the results for all three types of chassis are normalized to a sample size of 68, which is that of the A. Calculations are made on a daily and weekly basis, wherein 1 week equals 6 days, for 0.1, 0.01, and 0.001 probability levels.

Since the A and B chassis are a dual channel, *i.e.*, two amplifiers to each chassis, and the C a single channel amplifier, normalizing the C sample to the A sample required an additional factor of k. In the ideal case $k=2$, but $k>2$ presents a more realistic figure. This is so since two C chassis are equivalent to one A chassis; however, pulling out an A chassis for repair when only one channel has failed is harmful to the good channel, and increases its chance of failure upon being replaced.

From the data in Fig. 4(c) an approximate ratio of 1.3 of initial failures to the average number was obtained for A chassis, and this gave a figure of 2.6 for k.

Thus for a confidence level of 0.001, the A requires eight spares, the B, 12, and the C, 11 spares on a weekly basis.

There is another interesting point about the k factor. Since k represents the ratio of initial to chance failures, it could also represent the increase in required personnel so as to expect no down time when the equipment has initially been installed and is being debugged. Conversely, if one plans on using only those number of people initially who would be required ultimately during steady state operation of the equipment, he can expect a down time of the order of the k factor. One can then trade down time for people within the limits of the k factor.

Logistics Considerations

General

As a result of the failure analysis, a value is obtained for the spare parts replacement based upon the steady state performance of the equipments. In the illustrative problem, this is for computing amplifiers, and the value obtained is N spares per T time interval, at a confidence value of P per cent. Thus, if we are willing to accept a risk of P per cent, N spares per T time interval should enable us to operate full time within the stated risk level. This result, for spares, together with some simplified considerations, can be applied toward estimating logistics. For a base-field station example, a number of factors should be taken into account.

1) Average repair-time cycle per amplifier.
2) Shelf deterioration.
3) Equipment obsolescence.
4) Number of repair technicians.
5) Effect of transportation pipe line delay between field stations and base.

Average Repair-Time Cycle per Amplifier

It was estimated that on the average, one good technician could repair two A amplifiers per day.

Shelf Deterioration

The observed shelf deterioration of the B and C amplifiers is negligible over a six months basis. The A amplifiers use electrolytic capacitors, and so probably should not be kept as spares longer than six months at any one time. Since a rotating system is used in which spares replace faulty amplifiers, shelf deterioration should not be a factor.

Equipment Obsolescence

Eventually all equipment gets obsolete. In case of obsolescence, the amplifiers either can be replaced or else rebuilt. If too many spares are kept on hand and equipments have to be replaced the cost may be high. Hence, an optimum balance is deisrable between number of spares on hand and repair technicians.

Number of Repair Technicians

It may be desirable to have more than one repair technician, since they will be able to aid and abet each other. In addition, if one is absent, the work can still continue approximately on pace. This should be balanced with the required number of spares and magnitude of the facility.

Effect of Transportation Delay between Field Stations and Base

The delay, or pipe line between the base facility and the equipment in the field stations, means that additional amplifiers over and above the shelf spares are needed to fill the pipe line. The pipe-line delay should be kept as small as possible.

Example

Table VI shows the results of the spares determination on a daily basis for the three groups of amplifiers normalized to 68 chassis, *i.e.*, two amplifiers per chassis. If we consider the case of the A amplifiers, at a 1000:1 confidence level, it can be seen that four spare amplifiers per day will suffice. Suppose for purposes of illustration, we take the worst case for three days in a row. At the conclusion of the first day, assume four amplifiers go bad. Then assume at the start of the second day four additional amplifiers go bad. That means in order to be operating on the second day, eight spares will have been required initially. However, one repair technician can repair two amplifiers per day. Therefore, at the end of the second day we shall have two operable spare amplifiers back in stock.

On the third day, assume four more amplifiers go bad at the beginning of the day. Then we would need four spares in order to operate fully the third day. However, two spares are in stock after having been repaired so that only two additional amplifiers are needed. Thus, for the case mentioned, a total of $4+4+2=10$ spares plus one competent repair technician will be required to insure the station operates within a large confidence limit.

We can estimate this limit as follows. Our calculations show there is one chance in 1000 that more than four spares will be required per day. The probability that exactly four independent failures 1) will occur per day is given by $P(4)=16\times10^{-5}$ from the Poisson relationship, and 2) occur three days in a row is $(16\times10^{5})^{3}=16^{3}\times10^{-15}$. Thus, the occurrence of four failures three days in a row in the manner discussed represents an extreme.

If we assume we have two competent repair technicians, and the same failure situation as previously, then at the end of the second day we would have $2\times2=4$ amplifiers repaired, which amplifiers would be required for the third day. Thus, in this case, we need eight spares to carry us over the three-day period. It is advantageous in many cases to have two repair men, so that two repair men plus eight spares for the 68 A amplifier chassis would be preferable, as well as provide a big margin for safety.

An additional modification must be allowed for in the delay caused by the pipe line. Assume a field station at such a distance that there is a delay of one-half day between it and the base. This one-half day represents, on the average, the amount of time required for one man to repair one amplifier. Hence, in the case of the ten spare amplifiers plus one repair man, we would need one more spare to compensate for the pipe line delay time. Thus a total of eleven spare A amplifiers plus one technician would be required. In the case of two technicians plus eight spares, one-half day represents, on the average, the amount of time needed for two repair men to repair two amplifiers. Hence, two repair men plus ten spares would be required.

The case of three repair technicians is interesting. At the start of the second day, eight spares are needed. By the end of the second day, $3\times2=6$ amplifiers have been repaired, so that a sufficient number of spares are on hand at the start of the third day, with two amplifiers to spare. The pipe line will be $3\times2\times\frac{1}{2}=3$ amplifiers.

However, there are still two good spares available, so that three repair technicians plus $8+(3-2)=9$ amplifiers will be required.

It may be questioned as to whether such extreme confidence levels are required. In each case, the physical nature of the problem at hand should be examined carefully, and analytical results used as the best guide to determine appropriate parameters. It is interesting to note that we cannot replace spares by technicians directly, but must consider the factors involved. Doubling or tripling the number of technicians will not reduce the number of spares required in the same proportion.

The part played by the delay in the pipe line shows up very noticeably. To minimize these losses, as short a pipe line delay as possible should be the aim of the logistics pattern.

These few aspects of the base-field station example merely indicate one approach in which spares estimates can be used for setting up logistics procedures. More detailed analyses can be carried out for complete solution of some logistics system problems. It can be recognized that determination of a logistics scheme is similar to that of inventory control, and is another aspect of the more general waiting-line problem.

Bibliography

[1] Jeffreys, H. *Theory of Probability*, London: Oxford Press, 1948.
[2] Geisler, M. A., Brown, B. B., and Hixon, O. M. "Analysis of B-47 Consumption Data and Activity," Rand Corp. RM-1288, July, 2 1954.
[3] Brown, B. B., and Geisler, M. A. "Analysis of the Demand Patterns for B-47 Airframe Parts at Air Base Level," Rand Corp. RM-1297, July 27, 1954.
[4] Lundberg, O. "On Random Processes and Their Applications to Sickness and Accident Statistics," Uppsala, 1940.
[5] Berman, E. B. "A Model of the Procurement-Repair Decision for a Spare Item," Rand Corp. RM-1519, July 25, 1955.
[6] Berman, E. B., and Clark, A. J. "An Optimal Inventory Policy for a Military Organization," Rand Corp. P-647, March 30, 1955

Accuracy Control Systems for Magnetic-Core Memories

A. KATZ†, A. G. JONES†, AND G. REZEK†

Introduction

WHY BE concerned with means for improving reliability in magnetic-core memories? The coincident-current magnetic-core memory has proven to be the most reliable medium yet devised for high-speed storage in digital computers. Experience with core storage at BIZMAC, as well as at Lincoln Laboratory and at RAND,[1] shows that one can achieve mean times between errors measured in the hundreds of hours. Should this, however, be the basis for complacency? Such performance, although an order of magnitude better than has been reported[2] for electrostatic storage, is only comparable to those of the arithmetic and control portions of the computer.

It is our basic premise that core memory performance, while presently adequate, will not long suffice in view of the trend toward greater complexity resulting from increases in memory speeds and capacities. If this premise be accepted, then it follows that means must be provided for enhancing memory reliability. Two such means will shortly be described.

Accuracy Control

Reliability has been defined "as the probability of a device performing its purpose adequately for the period of time intended, under the operating conditions encountered."[3] In the case of a computer, a transient fault can invalidate the results of extensive computation. A useful measure of system reliability is, then, the probability of error-free operation during a given run as a function of the duration of that run.

The usefulness of the processed results is directly related to system reliability. In executing its instructions, the computer must accurately perform the many transfers and transformations whereby data is processed. Control of accuracy is facilitated by checking: programmed, built-in, or some combination thereof. The importance of checking and the extent to which it is applied depends on the consequences of improper operation. If these consequences are measurable, then the "best" combination may be determined on an economic basis. More frequently, however, the balance is established empirically, and consequently reflects the wide variance of opinion as to the "best" combination.

† Radio Corp. of America, Camden, N. J.
[1] RAND JOHNNIAC. See *Digital Comp. Newsletter*, vol. 8, April, 1956.
[2] J. M. Wier, "Reliability and characteristics of the Illiac electrostatic memory," *Proc. East. Joint Comp. Conf.*, pp. 72–77; 1953.
[3] L. M. Clement, "Reliability of military electronic equipment," *J. Brit. IRE*, vol. 16, pp. 488–495; September, 1956.

Magnetic-Core Memories

In the course of the past two decades there have been several generations of memory devices. Although many devices are capable of retaining binary information, relatively few lend themselves to rapid selection. One of these few, the bi-remanent magnetic core, has recently become the "standard" storage medium in high-speed computers. The individual core acts as an elementary cell capable of storing one binary digit. As shown in Fig. 1, the core stores a "0" when in positive remanence; "1," in negative remanence. The memory element is relatively insensitive to an applied field H_d, but is responsive to a field H_m resulting from the coincident application of two fields of value H_d. By virtue of this nonlinearity, the cores provide an added degree of discrimination which greatly simplifies the selection problem.

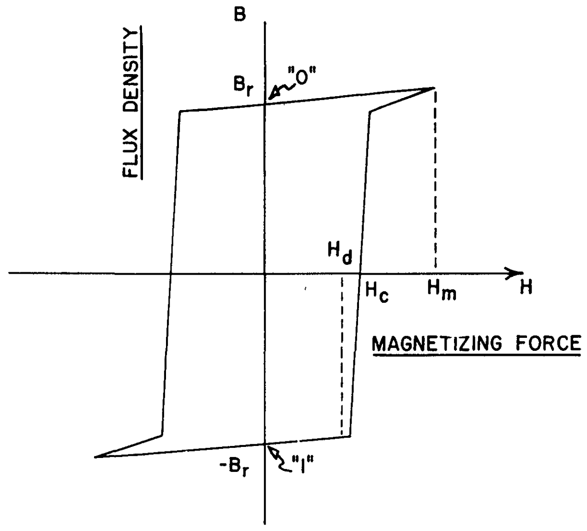


Fig. 1—Hysteresis characteristic of memory core.

Coincident-Current Operation[4–6]

The principle of operation will briefly be reviewed. In the array shown in Fig. 2, each core is threaded by four windings: the sense and the inhibit windings are common to all cores, a particular x coordinate and a y coordinate access line threads each core. The content of a specific core is read by applying a drive current I_d along each of the appropriate access line pair. Only the core at the intersection is driven by $I_m = 2I_d$, and only that core responds. If it contains a "1," a relatively large voltage is induced in the sense winding; if a "0," a relatively small voltage. In Fig. 2, the content of core 21 is being read by driving lines X_2 and Y_1.

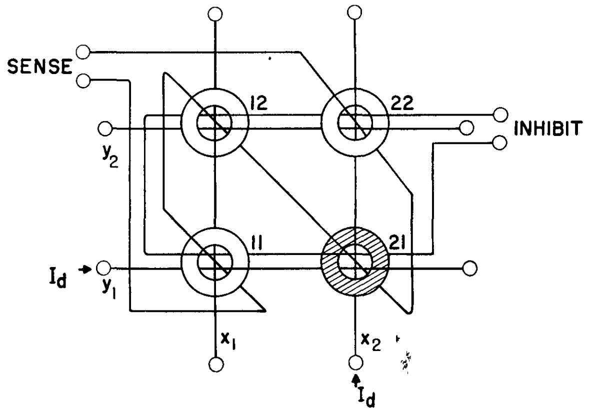


Fig. 2—Planar array of memory cores.

To write information into a core, the drive currents are reversed. If a "0" is to be inscribed, a bias current having magnitude I_d and sense opposite to the drive current is applied to the inhibit winding; if a "1," no excitation is applied. Note that the extraction of the content of a core results in destruction of the information in that core—whatever its original content, it will be "0" after reading. Hence, an access to the memory requires a "read-write" cycle if the information must be preserved for later use.

The planar arrays are arranged to form a compact, three-dimensional lattice by interconnecting corresponding x lines and y lines. Each array serves as a digit plane since it stores a particular digit for each of the registers in the lattice. Storage elements are selected by suitably controlling the currents in the three coordinates along the edges of the lattice. Since the selection coordinates are entirely spatial, the rate of access to any register is inherently high.

A block diagram of the basic memory configuration is shown in Fig. 3. A particular core register is selected by the X access and Y access means. Each of these consists of an address register, selection matrix, access drivers, and access switches. The information channel includes digit drivers, sensing amplifiers, strobe gates, pulse standardizers, a memory register, and a network for gating information to and from the memory. There is also a memory-timing generator which converts the basic machine commands into pulses of a nature determined by the characteristics of the cores.

Accuracy-Control Means

Having reviewed the basic ideas, let us now consider two systems for the detection and location of memory faults. The first of these determines if the desired register has been selected; the second, that the desired information has been inserted into the memory. In each case, the actual results are monitored and compared with the results desired. If the comparison fails, an alarm condition is indicated and the machine is stopped.

[4] J. W. Forrester, "Digital information storage in three dimensions using magnetic cores," *J. Appl. Phys.*, vol. 22, pp. 44–48; January, 1951.

[5] J. A. Rajchman, "A myriabit magnetic-core matrix memory," Proc. IRE, vol. 41, pp. 1407–1421; October, 1953.

[6] W. N. Papian, "New ferrite-core memory uses pulse transformers," *Electronics*, vol. 28, pp. 194–197; March, 1955.

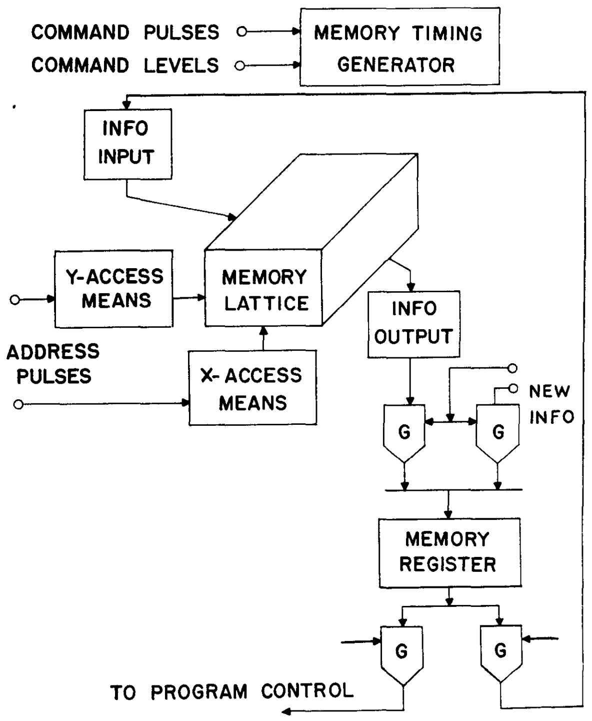


Fig. 3—Block diagram of core memory.

Checking of Register Selection

The block diagram of the core memory has been redrawn in Fig. 4 to show those areas pertinent to the selection of a core register. With each access system is now associated a magnetic-core error matrix, the outputs of which are fed to a logical network which determines if an alarm condition exists. The details of a check system for one access dimension (8 lines) are shown in Fig. 5.

The theory of operation is as follows: in the course of a memory cycle, the binary code for the desired line is set into the memory address register from a counter in the program-control portion of the computer. A coordinate line, hopefully the correct one, is then driven by one of the access drivers. Each line threads its way through the memory lattice, and then through the appropriate set of error cores (represented by the short diagonal lines) with six turns. Thus, if a current of value I_d produces a magnetizing force H_d in the memory cores, it will produce $6H_d$ in the error cores. Since the error cores are identical in characteristic with the memory cores, the appropriate set of error cores will be switched whenever a line is driven.

Examination of the error matrix shows it consists of two parts: the leftmost two columns of cores determine the parity of the line being addressed, while the rightmost three columns re-encode from linear to binary. The outputs of the parity-detecting cores are amplified and fed to a network which checks that at least one and no more than one line was driven during the memory cycle. The outputs of the address-encoding cores are also amplified and fed to a network where the address code

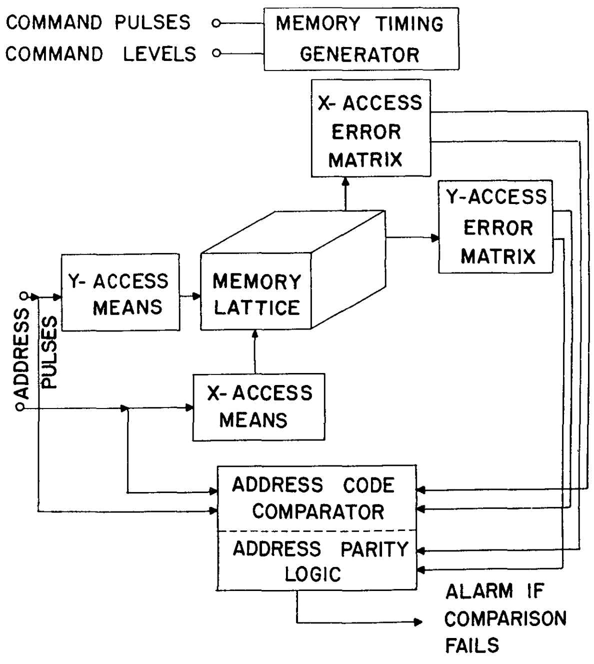


Fig. 4—Checking of register selection.

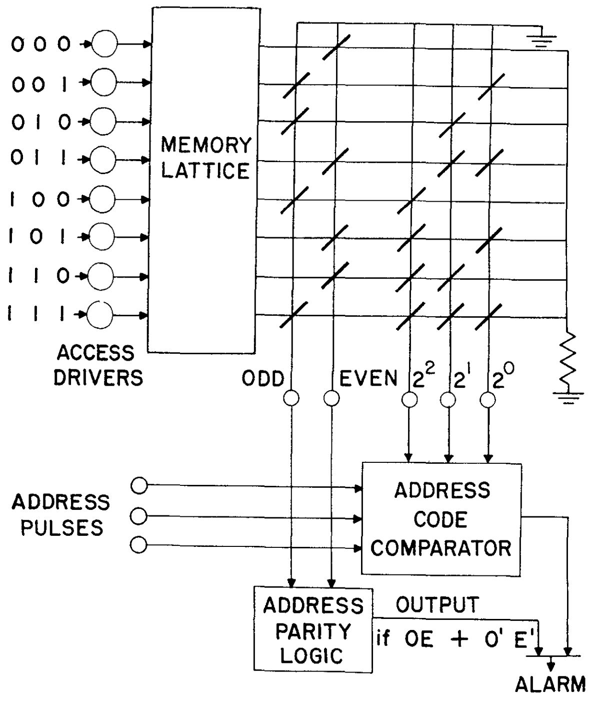


Fig. 5—Error-matrix and comparison logic for one access dimension.

sent by the originating counter is compared against that of the line driven. An alarm is indicated in case of malfunction.

The system just described will detect and locate faults which occur in register selection. Since the current in the access line is monitored, no link in the selection chain is permitted to function without scrutiny. The control loop is closed around the entire selection system.

If economy, space, or weight are the overriding considerations, then a simpler checking system might be

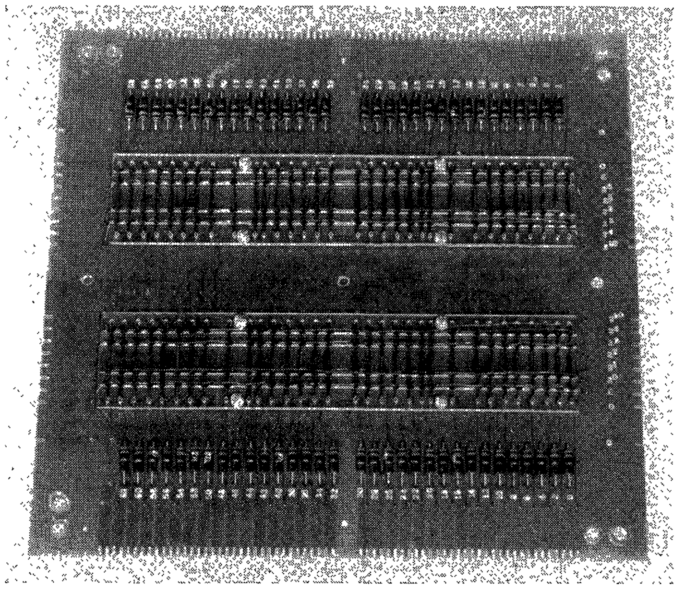


Fig. 6—Error matrix.

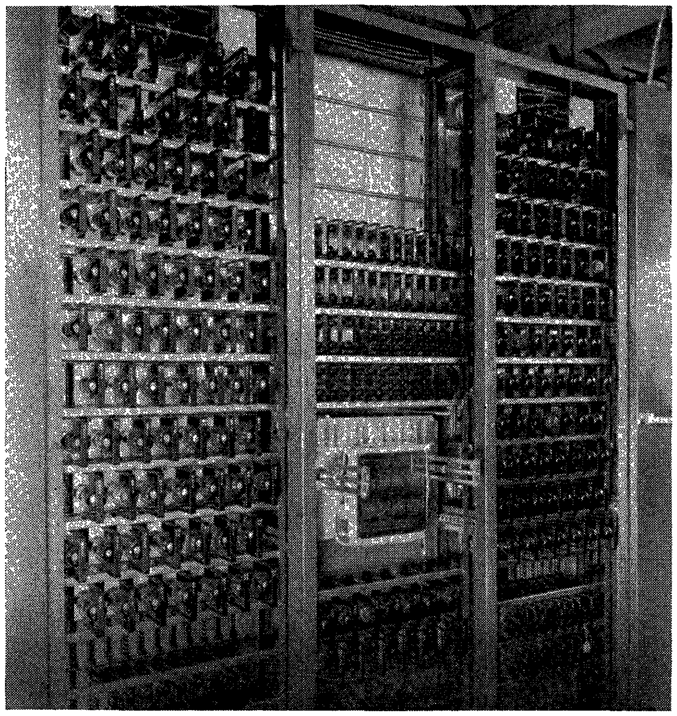


Fig. 7—Error matrix under test.

based on the parity-detecting cores. This system would detect single faults, but would be of less effectiveness in locating the fault.

A photograph of the error core matrix is shown in Fig. 6. Note that the design is such that it may readily be assembled with the memory arrays into a complete lattice. A photograph of the prototype error matrix under test is shown in Fig. 7. Fig. 8 shows a photograph of the error-core outputs from the sensing windings when the matrix is in operation. Condition I shows the outputs of the parity-detecting cores under proper addressing, which is addressing one line at a time. Condition II shows one form of improper addressing, which is addressing two lines simultaneously caused by an open diode in the address matrix.

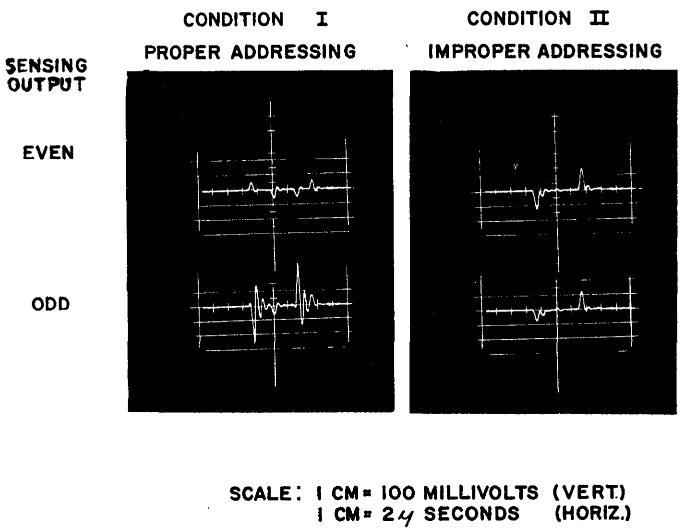


Fig. 8—Error-core outputs.

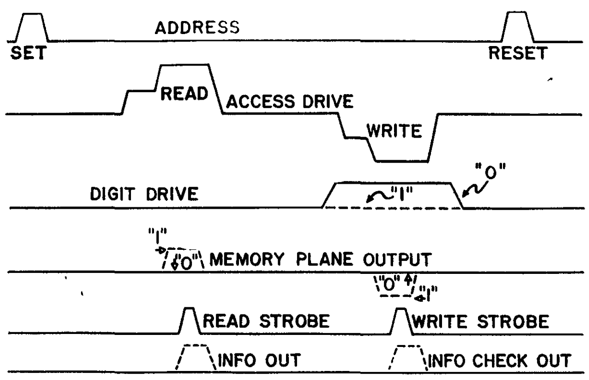


Fig. 9—Memory-timing diagram.

Checking of Information Insertion

Control of this memory function is extremely simple in concept. We recall that this memory operates on the principle of a "destructive read-out." Hence each memory access is characterized by a "read-write" cycle. If one examines the timing diagram shown in Fig. 9, one finds that an output is produced from each core in the selected register during both the "read" and the "write" portions of a memory cycle. For either portion, a relatively large output indicates storage of a "1"; a small output, a "0." Whereas the output at "read" time identifies the information which had been stored during an earlier memory cycle, that at "write" time identifies that just inserted in the core. By providing an additional strobe pulse, the core output at "write" time may be sampled and stored in a memory check register.

Referring to Fig. 10, one finds a simplified diagram showing the information channels with their associated accuracy-control logic. Since the memory register contains the information that should have been stored in the core register, and the check register that which actually was stored, a comparison can be made. In case of error, an alarm is indicated and the machine is stopped. Here again the control loop is closed around the entire function, since it is the final remanent state of the memory core that is being monitored.

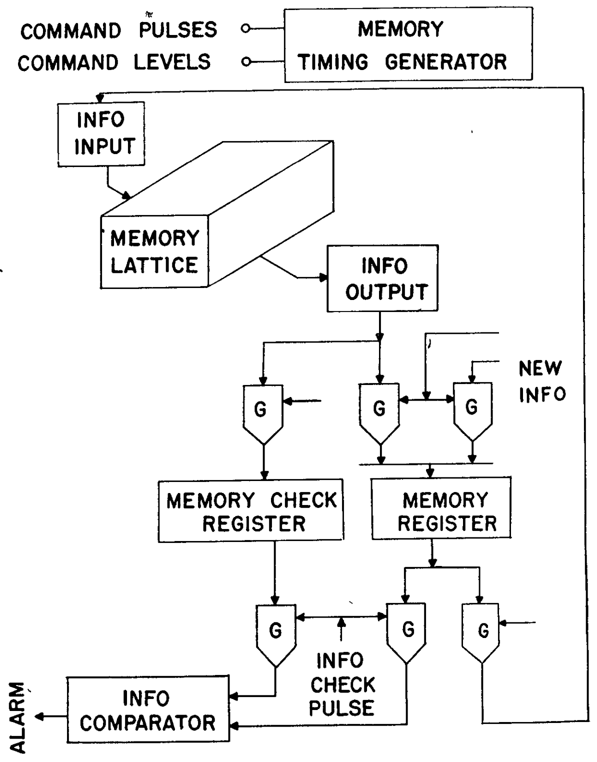


Fig. 10—Checking of information insertion.

Balance In Means for Checking

The provision of checking features, be they programmed or built-in, requires additional apparatus, increased machine working time, or both. These consequences must be weighed against those resulting from improper operation of the system. Furthermore, a balance must be struck between the relative proportions of programmed and built-in checking. The more built-in checking provided, the higher will be the initial cost. A hardware-checked machine, however, will provide savings in operation through reductions in the following:

1) programming costs,
2) program debugging costs,
3) key-punching costs,
4) computer running time per task,
5) storage requirements.

Where clearly desirable, hardware checks should be provided in a manner consistent with the system philosophy and with the needs of the particular application. The amount of additional apparatus required may be held to a minimum by designing around checking features inherent in the system and by integrating all such features on a system basis.

It should be noted that no amount of checking will, of itself, produce useful data. Implicit throughout has been the assumption that the computing system is the end product of mature circuit and system design, and of high standards of workmanship.

Conclusion

The importance of checking computer operations is generally accepted. Economic considerations stemming from consequences of improper operation dictate extent and means for implementing checks. A balanced combination of built-in and programmed checking appears to be adequate for commercial applications.

Two systems are discussed for improving the reliability of operation of a magnetic memory. These systems, simple in concept and economic in implementation, provide checking functions which are extremely difficult to accomplish by programmed means. By detecting and locating faults as they occur, these checking systems will raise the general level of performance of the memory. Improvement in performance is necessary if we are to keep pace with the continuing trend to higher capacity and speed in computer memories.

Discussion

E. J. Otis (Daystrom Systems): In Fig. 5, if lines such as 7 (111) is selected, and line 2 (010) is by mistake pulsed, how is the system going to recognize the error?

Dr. Katz: Line 7 consists of three ones, and line 5—line 2 (010) are both of odd parity, so that the parity detecting cores would not sense this error. However, the address and coding cores would encode 111, which is the proper address. From this we go right into the address code-comparator, and compare favorably with the address pulse. The parity-detecting cores, which I mentioned earlier, would not detect the error. This error, however, involves the failure of two bits; in other words the two to the zero, and the two to the second bit have failed, and such a simultaneous failure is very unlikely, but this would not have detected it. Now I should amplify it by saying that this system will detect all single-bit failures; seven out of eight 3-bit failures, etc. I did mention that although this system would detect virtually all errors, it will not detect this particular error.

John Paivinen (General Electric): Please repeat the description of how data check is performed. Is it read-out following a write operation, or are the inhibit drivers monitored?

Dr. Katz: Neither, actually. In the process of writing information into the core, if the core were to receive a one, it would be switched and go back to $-B_r$ in this region. If it were to have a zero inscribed, it would not switch and would be a relatively small output. The same sense findings would see both outputs, C_1 and read times, as well as C_1 and write time. And the same sense amplifier would amplify the signals, since the response is both positive and negative polarity signals. By providing an additional strobe, at a different time, one can examine the output of the plane at the time of writing. The same sensitive equipment that is involved in normal reading is again used in checking. We do not monitor the current in the inhibit drive.

Vaughn Winkler (IBM): You indicated 15–30 per cent more equipment was required for register selection. How much additional time was required for information channe checking?

Dr. Katz: That 15–30 per cent was pulled out of context. The 15–30 per cent check applies to the total system, in that it means an increase in equipment for the total system. Now, the two systems described here—the increase and the equipment associated with the computer—as far as these two checks were concerned, would only be about 5 to 10 per cent of the memory itself. It would increase the digits for the most elaborate address check, and the information check. Now the information check is a function of how many bits are in a word; and the address check depends on how many bits determine one dimensional of access.

Now insofar as additional time on the information, there is no additional time on the register selection check; this happens in the process of reading. With respect to the information check, there is possibly another half-microsecond cycle, which is a staggering of the write pulse, resulting in less noise at the write time. The strobe occurs at the write time.

Design of a Basic Computer Building Block

J. ALMAN,† P. PHIPPS,† AND D. WILSON†

Foreword

IN THE PAST, circuit design of a basic transistor building block consisted of design by successive approximations, where known operating circuits were improved upon by using laboratory techniques to wire up new circuits and then the working properties of these new circuits were evaluated by measurement. A technique for designing and regulating the characteristics of a new circuit with a minimum of laboratory verification is described. Most of the actual work is accomplished by a large scale digital computer, the Univac Scientific.

The use of the digital computer in these circuit designs makes possible a degree of circuit investigation which was hitherto impractical to perform because of time and manpower limitations in the laboratory. The Univac Scientific can, in a few hours, do years of circuit investigation.

Fig. 1 shows a team developing circuits. One member is a circuit engineer experienced in circuit development. The other member is a mathematician experienced in Univac Scientific operation.

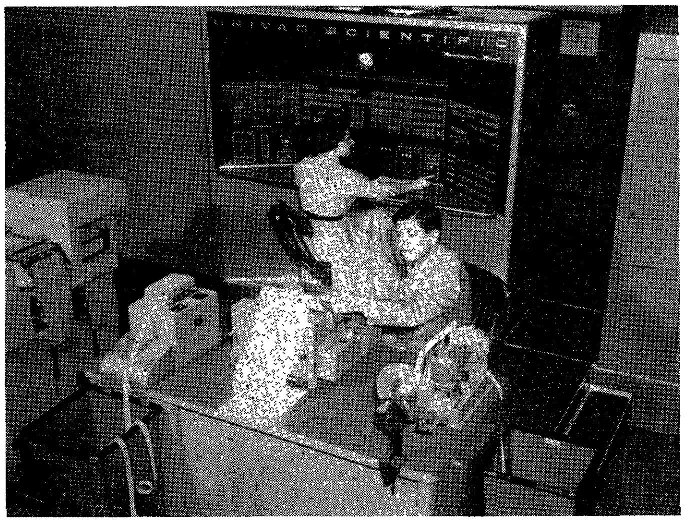


Fig. 1—Operating the Univac Scientific.

General

In this case, the goal was a type of transistor dc inverter circuit having several "or" inputs and several "and" outputs. The general design is shown in Fig. 2.

The computer proves a most valuable tool for the circuit design, once the circuit equations are established. Not only can the computer determine which circuit best meets the specifications, but it also is able to determine just how acceptable the optimum circuit is.

† Remington Rand UNIVAC, St. Paul, Minn.

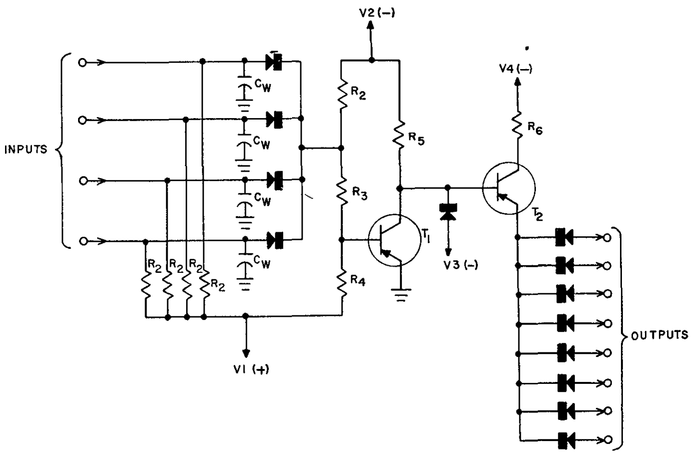


Fig. 2—Basic building block circuit.

The design is accomplished in two phases. Phase one consists of developing the circuit equations. The computer is then programmed to solve these equations while commuting the variables for two reasons: first, to find all possible solutions and to indicate all those combinations which are not solutions of the desired circuit, and second, to determine and print the particular solution which best meets the desired circuit specifications.

The input to the computer would be component specifications and desired circuit performance specifications. The computer then searches for the value of resistors that would meet these circuit specifications when all the components are in the worst possible end-of-life conditions.

Phase two consists of a series of equations which indicate if the circuit is operative with the values programmed into these equations. The program varies the circuit parameters about the nominal values for the "best" circuit computed in phase one. Failure points are then established for different values of the circuit parameters. Curves are plotted to show areas of circuit operation and areas of circuit failure.

Phase I—Developing the Equations

The equation of a circuit may be developed by several methods, one of which employs straightforward classical circuit theory analysis. A spot-check is maintained on the equations by inserting values of circuit parameters in the equations to determine actual operating conditions. Another method is that circuits may be set up and their operation noted, and from these observations, relationships can be observed to derive the equations necessary to express the circuit. A close check must be made in the laboratory to be certain that none of the approximations made in the development of these equations may have a detrimental effect on the final results.

The design equations for the transistor inverter are developed as follows:

The time constant equation for the over-all circuit is:

$$T = R_L C_W \tag{1}$$

where

T = time for capacitor to charge to 63 per cent of its final value
R_L = resistance of circuit in ohms
C_W = capacity of circuit wiring.

But, from Ohm's law,

$$R_L = E/I. \tag{2}$$

So that substitution of (2) in (1) produces the linear approximation for rise or fall time of the circuit:

$$T = EC_W/I. \tag{3}$$

This circuit is to be used in a digital machine and only two states of the circuit have to be considered. These two conditions occur at end-of-life operation and are defined as the most extreme voltage levels which can possibly appear on one output. These signal voltages are thus defined as E_0 and E_2, where E_0 is the signal voltage at the worst end-of-life case above ground, and E_2 is the signal voltage at the worst case as determined by V_3.

Substituting E_0 and E_2 in place of E in (3) and adding the expression I as the current in the load.

$$T = (E_0 - E_2)C_W R_L/V_1. \tag{4}$$

Eq. (4) now gives the first approximation of the speed of the circuit, T being the rise or fall time of the circuit (the time required to make the circuit switch from one state to another state of signal level).

Next, to determine the amount of power dissipated in the transistor designated as T_2 in the circuit, the power in this transistor can be defined as the current flowing through the transistor times the voltage across this transistor in the worst case, or:

$$P = NEI \tag{5}$$

or

$$P = -NV_4V_1/R_L \tag{6}$$

where

N = the number of outputs
V_4 = the voltage on the collector electrode
V_1 = the voltage tied to the load resistor
R_L = the resistance of one load.

The next design equation to be defined is for the value of R_5. The value of this resistor can be defined as

$$R_5 = E/I \tag{7}$$

where E is the voltage across this resistor in the worst case and I is the current through this resistor in the worst case, now; defining E as the voltage in the circuit,

$$E = (E_2 - V_2) \tag{8}$$

and defining I as the current necessary to drive the worst case load,

$$I = (V_1 - E_2)N/R_L B_{T2} \tag{9}$$

where B_{T2} is the current gain of the transistor designated as T_2. By subsitution of (8) and (9) into (7), a direct relationship to determine R_5 is found:

$$R_5 = B_{T2}(E_2 - V_2)R_L/N(V_1 - E_2). \tag{10}$$

Next, it is necessary to derive the necessary current input to transistor T_1 which would be necessary to propagate the signal to the output of T_2. This current can be defined as I_D. In the worst case, the input current I_D is equal to

$$I_D = V_2/B_{T1}R_5 \tag{11}$$

where B_{T1} is the current gain of the transistor designated as T_1. To determine the value of R_3 and R_4, a consideration of the end-of-life conditions of the circuit is made. For the E_0 condition where E_0 is the end-of-life value for the signal voltage above ground

$$I_5 = I_7 - I_{C0} \tag{12}$$

where I_5 and I_7 are currents as designated on the circuit diagram and I_{C0} is the cutoff leakage current of transistor T_1.

I_5 and I_7 may be defined as functions of the circuit parameters as

$$I_5 = -E_0Y_3 \tag{13}$$

where

$$Y_3 = 1/R_3 \text{ (the conductance of } R_3) \tag{14}$$

and

$$I_7 = V_1Y_4 \tag{15}$$

where

$$Y = 1/R_4 \text{ (the conductance of } R_4). \tag{16}$$

After substitution of the values for I_5 and I_7 from (13) and (15) into (12),

$$-E_0Y_3 = V_1Y_4 - I_{C0} \tag{17}$$

it should be noted that (17) has variables of E_0 and I_{C0}, but the amount of drive necessary to excite T_1 is also of interest. This may be defined as I_D, the current necessary to supply the base with sufficient drive to operate the circuit. Solving for the value of R_3 and R_4 which will then satisfy both binary conditions of the circuit for the E_2 conditions proceeds as follows:

$$I_5 = I_7 + I_D \tag{18}$$

and

$$I_5 = -E_2Y_3 \tag{19}$$

where E_2 is the value of a binary signal. Also:

$$I_7 = V_1Y_4. \tag{20}$$

So substitution of (19) and (20) in (18) gives

$$-E_2Y_3 = V_1Y_4 + I_D. \tag{21}$$

Eqs. (14), (16), (17), and (21) may be solved simultaneously, and it is found that

$$R_3 = E_0 - E_2/(I_{C0} + I_D) \tag{22}$$

and

$$R_4 = V_1/(-E_0Y_3 + I_{C0}). \tag{23}$$

We solve for the value of R_2, the resistor which is between the input diodes,

$$R_2 = E_2 - V_2/(I_D + I_7). \tag{24}$$

Since

$$I_7 = V_1R_4 \tag{25}$$

substitution of (25) in (24) yields:

$$R_2 = E_2 - V_2/(I_D + V_1/R_4). \tag{26}$$

The equations developed from (1) to (26) are those which express the absolute circuit conditions which must be met with no circuit supply voltage variation or no resistor change from designed resistance. As there must be an allowance made for changes in resistance and changes in voltages new variables must be added to the equations. These may be defined as

$\Delta = +$Resistance tolerance
$\delta = -$Resistance tolerance
$G = +$Voltage tolerance
and $H = -$Voltage tolerance.

A series of equations may now be set up which will allow the Univac Scientific to solve and determine what limits may be placed on such parameters as rise and fall time, transistor current gains, values of I_{C0}, etc. This series is called the programmed equations.

Programmed Equations

Step 1 $R_L = +NV_4V_2G^2/P_M\delta$ (27)
Step 2 Pick next smaller RTMA value $= R_{LT}$
Step 3 $P_N = NV_4V_2/R_{LT}$ (28)
Step 4 $P_M = P_NG^2/\delta$ (29)
Step 5 $R_S = R_{LT}B_{T2}\delta/KN\Delta$ (30)
Step 6 K starts at 1 and is incremented by 0.1 step in the loop to a maximum of 2
Step 7 Pick next smaller RTMA value $= R_{ST}$
Step 8 $I_D = -V_2G/B_{T1}R_{ST}\delta$ (31)
Step 9 $R_3 = (E_0\Delta/\delta^2 - E_2/\Delta)/(I_{C0}\Delta/\delta + I_D)$ (32)
Step 10 Pick next smaller RTMA value $= R_{3T}$
Step 11 $R_4 = V_1H/(E_0\Delta/R_{3T}\delta + I_{C0}\Delta)$ (33)
Step 12 Pick next smaller RTMA value: R_{YT}
Step 13 $R_2 = (E_2 - V_2)H/(I_p\Delta + V_1H\Delta/R_{4T}\delta)$ (34)
Step 14 Pick next smaller RTMA value R_{2T}
Step 15 $T_R = (E_0' - E_2')C_W/[V_1(1/R_{LT} + 1/R_{4T})H/\Delta + V_2H/R_{2T}\delta]$ (35)
Step 16 $F_T = (E_0' - E_2')C_W/[(-V_1H/R_{LT}\delta) - (V_2B_{T2}H/NR_{ST}\Delta)]$ (36)
Step 17 Check value of (36) against value of (35). If (36) is larger than (35), go back to (30) and increase K to the next larger value. If (35) is larger than (36), print answer. In any case, do not increase K past 2.

Note: T_F and T_R, from (35) and (36) were derived by making the linear approximation for rise and fall time while taking into account the effect of loading, where one inverter was driving N other inverters.

The programmed equations now make possible the computation of many circuits with each having different end-of-life limits, transistor gains, supply voltages, wiring capacity, signal levels, etc. With all the pertinent parameters commuted and different circuits computed, it becomes a simple task to pick the optimum circuit for a given set of specifications. Fig. 3 shows a sample format used as output of the computer to indicate the necessary parameters and quantities to meet specifications.

Format Key

B_{T1}	B_{T2}	N	Δ	δ
P_M	P_N	K	I_D	
T_R	T_F	I_{c0}	R_{LT}	
R_{2T}	R_{3T}	R_{4T}	R_{5T}	

Sample format output

$+6.0000$	$+2.0000\times10^1$	$+3.0000$	$+1.1000$	$+9.0000\times10^1$
$+1.17150\times10^{-2}$	$+1.0000\times10^{-2}$	$+1.3000$	$+3.1421\times10^{-2}$	
$+4.0000\times10^{-6}$	$+6.5710\times10^{-6}$	$+3.0000\times10^{-5}$	$+2.0000\times10^{+4}$	
$+8.2000\times10^{+4}$	$+5.6000\times10^{+3}$	$+1.8000\times10^{+5}$	$+6.2000\times10^{+4}$	

Fig. 3—Sample format.

Phase II—Checking

Once a circuit has been chosen which can meet the desired design standards, a new set of equations are developed which are called the checking equations. These checking equations check circuit operation with a change in circuit components. If a resistor changes or if a resistor and voltage input change simultaneously, these checking equations will indicate just how much the circuit elements may change with the circuit still meeting the desired design standards.

The checking equations are as follows:

$$-(E_2 + V_1/R_3R_4)B_{T1}R_5 + V_2 = 0 \text{ or } + \text{ (fails-)} \tag{37}$$

$$E_0 + R_3(V_1/R_4 - I_{C0}) = 0 \text{ or } + \text{ (fails-)} \tag{38}$$

$$(E_2 - Y_2)B_{T2}/R_S - (V_1 - E_2)N/R_L = 0 \text{ or } + \text{ (fails-)} \tag{39}$$

$$R_T - (E_0' - E_2')C_W/(V_1/R_{LT} + 1/R_{4T} + V_2/R_{2T}) = 0 \text{ or } + \text{ (fails-)} \tag{40}$$

$$F_T + (E_0' - E_2')C_W/(-V_1/R_{LT} - V_2/B_{T2}/NR_{5T}) = 0 \text{ or } + \text{ (fails-)}. \tag{41}$$

Eq. (37) checks the current in the collector circuit of transistor T_1 to determine if it is sufficient to maintain the dc level required by the circuit.

Eq. (38) checks the circuit to determine if the leakage current in transistor T_1 will not degrade the circuit operation.

Eq. (39) checks the current requirements of transistor T_2 to determine if it has enough current output to drive the required output loads.

Eqs. (40) and (41) check rise time and fall time respectively. These equations determine if variations in circuit parameters will increase the rise or fall time so that circuit specifications are violated.

With these checking equations, it is now possible to find the failure points for the circuit, and these points may be plotted in individual curves. The variation in one component is plotted against variation in another. This curve would now show which circuit elements are most critical. This evaluation now points the way towards a revision of the original equations to obtain a circuit which is even more impervious to component variations.

Fig. 4 shows a component curve developed about one parameter, R_L, all others being varied about their nominal value. All positions inside this curve denote circuit operation.

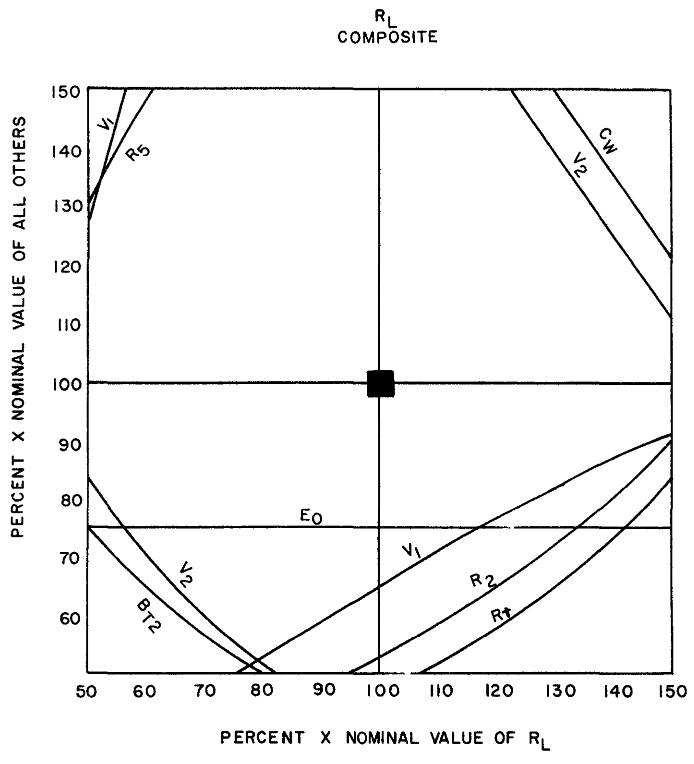


Fig. 4—A component curve.

Description of the Program for Phase I

The first part of the program, which is passed through only once in the run, forms the various differences and products that would remain constant throughout the run for the particular circuit under consideration. The "sub-setup" program then combines these results with the parameters to be commuted, *i.e.*, the transistor current gains, the number of inputs, and the resistance tolerances. The "sub-setup" is used wherever one of these parameters is commuted. The initial setup and the sub-setup programs shorten the actual calculation of the programmed equations a great deal. It then remains to run through the equations with various values of P_m, maximum power in the emitter-follower transistor.

The selection of the RTMA values from a table is carried out in the following way. The calculated resistance value is divided by the next smaller power of ten to bring it into the range of the stored table itself. After locating the next smaller tabular value, it is multiplied by the original power of ten. Fig. 5 (next page) shows an over-all organization of this phase.

For a typical program there are ten values of P_m, seven pairs of values of Δ and δ, four values of N, and five pairs of values of B_{T1} and B_{T2}. This results in 1400 different circuits to be analyzed. The calculation, which takes 3 hours of computing and output time, is done in floating-point arithmetic for ease of programming.

Description of the Program for Phase II

In the list of the circuit parameters, the program considers the first variables as the dependent variable and the second (then the third, then the fourth, etc. successively) as the independent variable, hereafter referred to as Y and X_i, respectively. For the nominal value of Y there is a test to see if there is a failure at 150 per cent of the nominal value of X_i and if so, X_i is increased in 5 per cent steps and failure is tested for until it finally occurs. These failure values are recorded and the X_i variable is tested at 50 per cent of nominal value for failure. If it fails again, X_i is decreased to the failure point and the results recorded. Otherwise, if there is no failure no changes are tried. After both sides of X_i have been tested for a given Y, the Y value is increased by 5 per cent and is analyzed as before. This is continued until the Y variable either fails on a nominal value of X_i or 150 per cent of the nominal of Y is reached. In either case the same procedure is carried out for decreasing Y by 5 per cent increments until, again, either a failure on nominal value of X_i occurs or 50 per cent of Y is attained.

At this stage the variable X_i and Y can be interchanged and the same analysis carried out. This interchange enables the program to follow the lines of failure completely across the 50–150 per cent interval considered, in most cases, although the calculation and the output time is almost doubled. However, there is surprisingly little duplication done due to this interchange. The variables are again interchanged to their original positions before a new independent variable is selected.

After one pair of variables has been completely analyzed, a new independent variable is taken and again the analysis is carried out. When all possible independent variables have been used for an initial Y, a new dependent variable is chosen and the set of computations for that dependent variable is accomplished for the remaining independent variables. See Fig. 6 for the program organization of Phase II.

In the given problem there are 15 variables and therefore 105 distinct pairs to consider. The calculation time is about one minute per pair of the average including output time. One may then expect the problem to run almost two hours before completion.

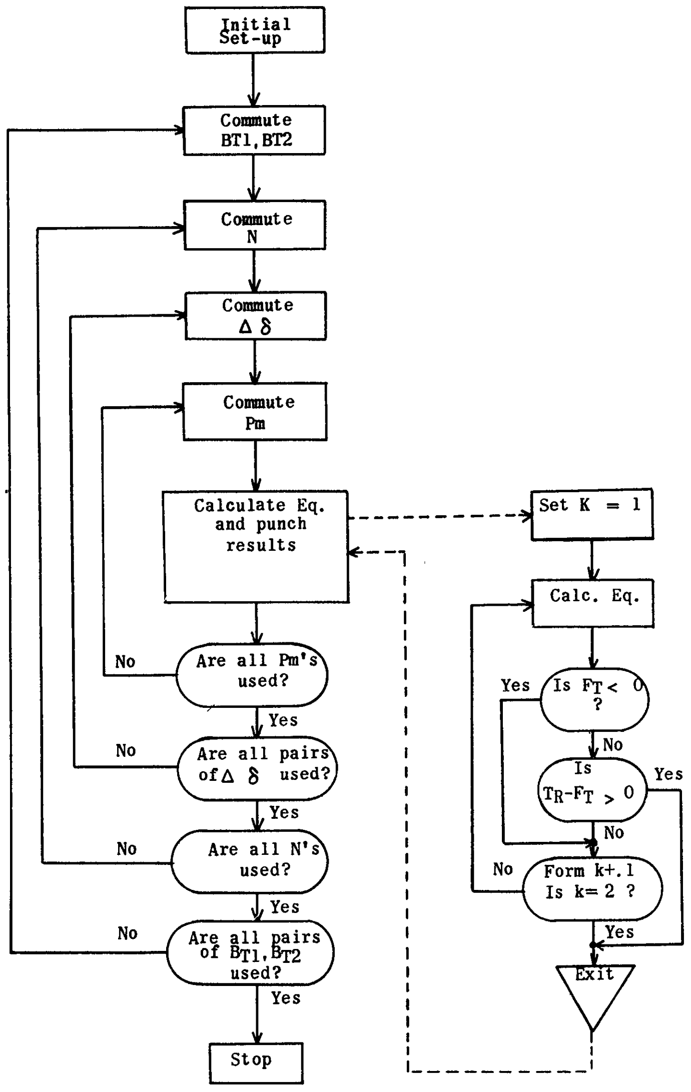


Fig. 5—Flow diagram for the first phase.

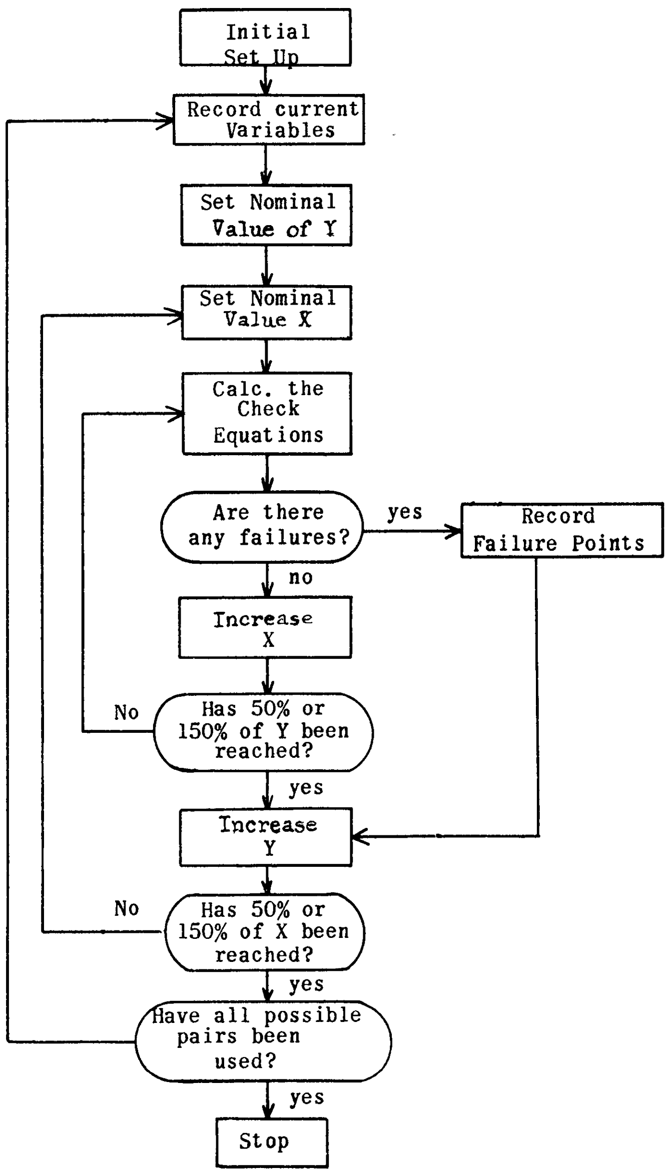


Fig. 6—Flow diagram for the second phase.

Conclusion

The method of design discussed here determines a way to utilize the Univac Scientific to do the detail work in developing circuits. It necessitates a minimum of engineering time to fully explore the circuit possibilities. As detailed an analysis which may be obtained with the Univac Scientific would either take years of laboratory work to obtain, or would not even be considered. Not only is the computer used to optimize the circuit, but it is also used to determine how well this optimum circuit will operate.

Discussion

Mr. Alman: The best circuit here is selected after the computer has calculated as many of the circuits as the designer feels might be of value; and the circuit designer then looks to the tabulated forms, plots the characteristics of these circuits, and picks the circuit which he feels has the widest power for components, and will best meet the circuit requirements.

Gunther Machol (IBM): How much time is required to produce the solution you have shown?

Mr. Alman: Six or seven hours to design the circuit, and about the same time for checking it. Of course, the minute you start doing this sort of thing you realize how many more circuits you might like to look into, and how much you might expand. We had one system set up which would take us something like three months to compute, so you can get into this pretty deeply. I would like to add at this time that this work was all done on the UNIVAC Scientific.

John Paivinen (General Electric): Why does not the transistor cut-off frequency appear in the rise-time relations?

Mr. Alman: The analysis we have shown has been simplified as much as possible. We have expanded the equation to take this fact into account. The drive current is known as is the reaction of the transistor to this drive current; this just adds an extra time on to the rise-and-fall equation.

T. P. Holloran (National Cash Register, Dayton): How long did formulation programming, trouble-shooting program take?

Mr. Alman: We had three people working on this development, and it took in the neighborhood of about three months to develop one circuit. Once we had the program going well, we could put a new circuit together in about a week, put it through the computer, and get results.

Error Detection in Redundant Systems

S. SCHNEIDER† AND D. H. WAGNER†

INTRODUCTION

IT IS WELL KNOWN that redundancy is a potentially powerful means of increasing reliability, provided the increased size, weight, maintenance, and cost are acceptable. A simple means of implementing redundancy is to rely on human detection of errors and subsequent switching to a standby unit. However, in control computers and other real-time devices, uninterrupted operation becomes crucial, and *automatic detection of error followed by automatic switching is thus required.* Indeed, this requirement would seem to be the heart of the problem of implementing real-time redundancy and is the problem to which this paper is addressed.

Investigations into two general areas are reported. In the first area, low-level duplication (*i.e.*, at the level of a single part or elementary circuit), the results are discussed only briefly, since they are largely negative under the rather stringent requirements which seem to be imposed by practical considerations. More positive results are presented in the second area, triplication with voting. Two practical forms of voting comparators are proposed, and one is quantitatively evaluated, taking account of both permanent failures and intermittence.

LOW-LEVEL DUPLICATION

Probably the simplest example of redundancy with automatic error detection and switching is a quad of diodes performing the function of a single diode (see Creveling[1]). Such a device counters a single failure (short or open) simply by automatically obviating the failed part. This technique is limited to rather specialized component functions, but applications other than to diodes exist. For example, if a capacitor is required merely to maintain at least a certain minimum capacitance, it may readily be replaced by a pair or quad of capacitors. Various configurations are shown in Fig. 1, together with evaluations based on Vitro failure data on electrolytic capacitors.[2]

Aside from the requirement that wide swings in parameter values be permitted, the quad technique appears to be seriously limited from the standpoint of maintainability. In circuitry built on quads, failure of a single part does not interrupt operation, but it does increase vulnerability to failure of its partner(s). As such failures build up, the reliability of the circuit decreases, eventually falling below the reliability without redundancy. The problem of locating such noncrippling failures in a large-scale machine appears formidable indeed.

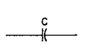
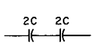
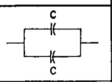
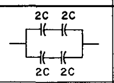
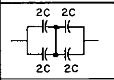
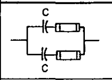
I	II	III	IV	V	VI
					
NO PROTECTION	PROTECTION AGAINST SHORTS ONLY	PROTECTION AGAINST OPENS ONLY	PROTECTION AGAINST OPENS AND SHORTS (CHIEFLY SHORTS)	PROTECTION AGAINST OPENS AND SHORTS (CHIEFLY OPENS)	PROTECTION AGAINST OPENS AND SHORTS (EQUAL)
$P_F = P_S + P_O$	$P_F \approx 2 P_O$	$P_F \approx 2 P_S$	$P_F \approx 4 P_O^2 + 2 P_S^2$	$P_F \approx 4 P_S^2 + 2 P_O^2$	$P_F = (P_S + P_O)^2$ (PERFECT FUSES)
P_F IS THE CONFIGURATION FRACTION FAILING PER YEAR FAILURE OCCURS WHEN CONFIGURATION CAPACITANCE FALLS BELOW C $P_O + P_S \ll 1$ GIVEN $\begin{cases} P_S \approx 0046 = \text{SHORT PROBABILITY} \\ P_O \approx 0020 = \text{OPEN PROBABILITY} \end{cases}$					
$P_F \approx 0066$	$P_F \approx 0040$	$P_F \approx 0092$	$P_F \approx 0006$	$P_F \approx 0009$	$P_F \approx 0004$

Fig. 1—Comparison of various capacitor configurations.

Let us consider a preventive-maintenance concept called forced-failure checking as a method of overcoming this maintainability obstacle, which is probably also present in other low-level redundancy techniques. The idea is to check by use of an overload signal that can be borne by a pair, quad, etc., if *no* member has failed, but which causes failure of all partners of parts that have failed during operation. Such a scheme would permit a circuit to be renewed to perfect condition periodically and would be extremely effective. Unfortunately, parts with the necessary properties are not known.

Because of the attractive advantages of forced-failure checking and the feeling that some such technique is necessary to apply low-level redundancy, attempts were made to devise a low-level switch which would give the duplicated components the necessary properties. Fuses, thermistors, ferromagnets, the Hall Effect, and organic liquids were considered to form limit-type devices for this purpose, as they are generally adaptable to implementation of forced-failure checking. However, these devices suffer from slow reaction times, and at best offer protection in one direction only, *e.g.*, against failures such as shorts which result in overloads. Logic elements composed of units such as flip-flops and gates could be arranged to perform the switching function. However, with only two parallel outputs to choose from, differences cannot, in general, be resolved to determine which output is in error.

It certainly cannot be said that all possibilities for solutions to these problems have been exhausted; for example, arithmetic self-checking may be attractive.

† Burroughs Corp., Paoli, Pa.

[1] C. J. Creveling, "Increasing the reliability of electronic equipment by the use of redundant circuits," PROC. IRE, vol. 44, pp. 509–515; April, 1956.

[2] J. H. Lancor, Jr., "Parts Failure Analysis: Electronic Equipment Reliability Program," Tech. Rep. No. 25, Vitro Corp. of America, Silver Spring, Md.; 1951.

However, it seems to the authors that these kinds of difficulties will generally be encountered in attempts to carry out low-level duplication in practice. Attempts to devise such means have on two separate occasions led in a natural fashion to the design of comparators to take a vote on the outputs of triplicated systems and pass a majority signal. These techniques are discussed in the next section.

Voting Comparator Techniques

Figs. 2 and 4 consist of logical diagrams of comparator circuits, either of which can be used to pass a signal which is the majority of the outputs of A, B, and C. Fig. 3 shows a transistorized realization of Fig. 2, using direct-coupled transistor logic. The method of Fig. 4 may be readily transistorized, also.

The operation of these two methods is described in this section. The method of Fig. 2 is confined to binary applications, but the alternative method permits analog application, as well. Alarm and diagnostic functions are performed in both methods in addition to voting.

Method of Fig. 2

Referring to Fig. 2, each branch receives the same inputs, performs the same function, and, if no fault exists, produces the same output to the flip-flop. The state of the gate is then checked by a pulse of the input P_2, which succeeds in arriving at f, g, and h, according to the following propositions:

$$f = ABC \vee AB\overline{C} \vee A\overline{B}C \vee \overline{A}BC$$

$$g = A\overline{B}\overline{C} \vee \overline{A}B\overline{C} \vee \overline{A}\overline{B}C \vee AB\overline{C} \vee A\overline{B}C \vee \overline{A}BC$$

$$h = A\overline{B}\overline{C} \vee \overline{A}B\overline{C} \vee \overline{A}\overline{B}C$$

Here "A" means that the output from A was a pulse, "$\overline{A}$" no pulse, etc. It is seen accordingly that

1) The output at f is the same as the majority of A, B, and C and should be passed to the next stage, if any.
2) An output is obtained at h if, and only if, there is disagreement among A, B, and C. This is passed to an alarm signal without necessarily stopping the machine.
3) If, during a diagnostic period, A, B, and C are operated two at a time in rotation, the output at h can be used to determine which of the three has failed. This can also be done during operation by observing which of the six outputs at g indicates an alarm.

Thus the comparator performs the functions of vote-taking, alarm in case of disagreement, and diagnosis.

It may be seen in Fig. 2 that the outputs "g" pass through a filter to some form of error indicator. The function of this filter is to provide an output only when the error rate exceeds a predetermined figure. In the event of such a failure indication, the machine operator may, by using switch S_1, switch the operation from the two-out-of-three mode to nonredundant operation, taking the output directly from one of A, B, or C.

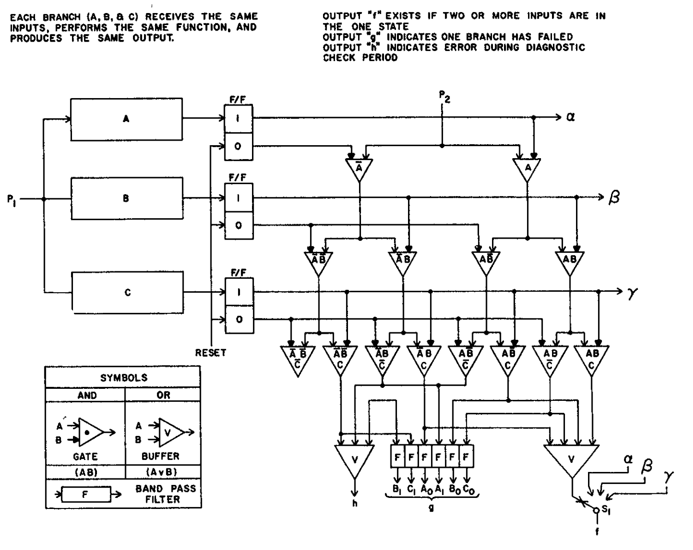

EACH BRANCH (A, B, & C) RECEIVES THE SAME INPUTS, PERFORMS THE SAME FUNCTION, AND PRODUCES THE SAME OUTPUT.
OUTPUT "f" EXISTS IF TWO OR MORE INPUTS ARE IN THE ONE STATE
OUTPUT "g" INDICATES ONE BRANCH HAS FAILED
OUTPUT "h" INDICATES ERROR DURING DIAGNOSTIC CHECK PERIOD
A
B
C
F/F
P1
P2
RESET
α
β
γ
SYMBOLS
AND
OR
GATE (AB)
BUFFER (AvB)
BAND PASS FILTER
h
g
f
S1

Fig. 2—Logical diagram of voting comparator.

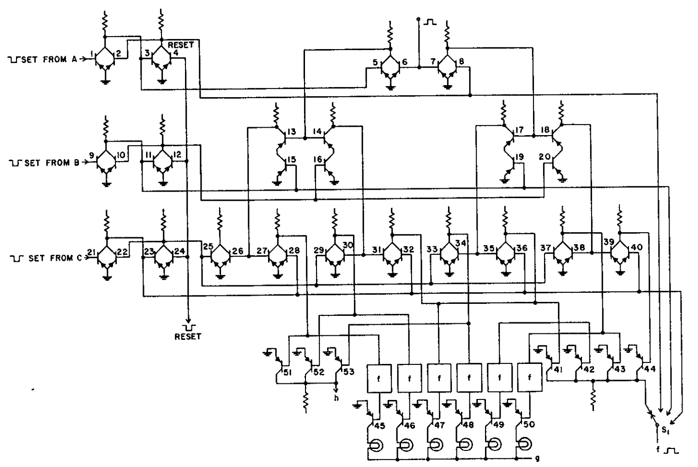


Fig. 3—DCTL mechanization of comparator.

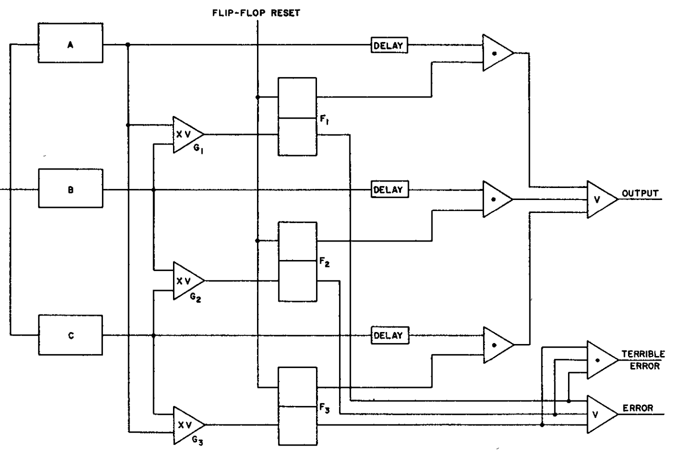


Fig. 4—Logical diagram of alternate voting comparator.

If the provision of the functions "*g*" and "*h*" is not required, then the mechanization can be reduced to a comparator of six gates. This still fulfills the functions of 1) operating correctly in the event of one branch failure, and 2) being capable of indicating the fault during the check period.

Referring to the DCTL version in Fig. 3, it will be noted that the configuration of the second stage is different from that of stages 1 and 3. This is required by the fact that when a signal goes through a stage there is a polarity inversion of the transmitted pulse. The circuit as shown does not take into account the need for additional transistors at junctions where the indicated numbers of connections may exceed the limits of DCTL design criteria.

Method of Fig. 4

The comparator shown in Fig. 4 performs the same function, but has a more general circuit which may permit its use with analog or binary, provided the signals are synchronized. The comparison is a circular one, in which the output of each component is compared directly to that of the other two. For example, let component A be putting a signal different from the signals of components B and C, and let the latter two agree. Then a signal will be passed by the exclusive-OR (XV) gates G_1 and G_3, whose function is to pass a signal only if the two incoming signals are of different level. The outputs passed by the gates go to F_1 and F_3 respectively, upsetting the flip-flops and deactivating the AND gates through which the signal must pass. A delay in the signal line prevents the wrong signal from getting through before the gate is deactivated. Thus the signals from A and C are cut off, but the signal from B passes through unhindered. The error signal given by the upset flip-flops can be used to initiate the reset of the flip-flops. The "terrible error" output is required to show that all three flip-flops have indicated an error and, therefore, that the output has been cut off, and is not just ZERO.

Comparison of Figs. 2 and 4

The difference between the methods of Figs. 2 and 4 is the manner in which the comparison is made. For the latter design, no signal need pass through the comparator mechanism unless an error has been made. On the other hand, the signals must be synchronized. This requirement differs from the method of Fig. 2, where the comparator is an integral part of the transmission network but absolute synchronization is not necessary. The difference between the two can best be summed up by saying that the comparator of Fig. 2 operates on the total signal output of the component, whereas the comparator proposed in Fig. 4 operates on the difference in signals between two components. This enables the latter to compare continuous as well as pulse signals, provided the comparator is suitably mechanized.

Evaluation of Comparator (Fig. 3)

In this section we discuss an evaluation of the comparator circuit of Fig. 3. If permanent failures only are considered, derivation of the reliability formula would be fairly easy. However, incorporation of intermittence into the formula becomes extremely complicated, so that recourse is made to simplifying approximations which generally yield a pessimistic estimate of the reliability of the triplicated system.

It is assumed that the computer is required to operate for a time t without failure (but not necessarily without error). The figure of merit used to evaluate triplication is the ratio of probability that an untriplicated circuit will fail before time t to the corresponding probability for the triplicated system. This ratio is defined as *gain*.

It has been shown by Cohn[3] that reliability ordinarily can be improved beyond that of the configurations considered here by using three comparators, each taking a vote and providing an input to one of three parallel branches in the next stage of computation. If the vote is being taken on the final stage, a fourth comparator presumably is added to vote on the output votes already taken.

Failures in the Comparator

Let us consider the consequences of various types of single failures in the comparator. Focussing attention on what happens during a single computational cycle, the situation is described by Table I, (next page), which tabulates the effect of shorts and opens in each individual transistor (according to the transistor enumeration in Fig. 3) under the assumption that everything else in the comparator is perfect. Columns I and VIII list the instances of single failure which will produce error even though the three inputs are all correct. Columns II to VII consist of error-producing failures when one of the three inputs is in error. Comparator failures that are not tabulated do not of themselves produce error, and are ignored.

The shorts are presumed to be collector-emitter, but opens can occur in the base, collector, or emitter. We assume that shorts and opens are equally probable. Then the rate of failures in the category of Columns I and VIII is six times that of a single transistor, while in taking all columns the multiple is 30.

Comparator Cutout

It can be shown that if an input circuit A, B, or C fails permanently in such a way that half of the subsequent bits will be in error, and if the comparator is perfect, then on each bit the error probability of the triplicated system is precisely the same as that of a single input circuit. Possibilities of failure in the comparator would then shift the balance in favor of no triplication. It is

[3] M. Cohn, "Redundancy in complex computers," *Proc. National Conference on Aeronautical Electronics*, pp. 231–235; May, 1956.

TABLE 1
COMPARATOR-TRANSISTOR FAILURE CHART

Transistor Designation (Fig. 3)	Signal (*ABC*) to Comparator							
	I 000	II 001	III 010	IV 100	V 110	VI 101	VII 011	VIII 111
1		−	−		+	+		
2		−	−		+	+		
3		+	+		−	−		
4		+	+		−	−		
5				+				
6				+				
7	+	+	+					
8	+	+	+		−	−		
9		−		−	+		+	
10		−		−	+		+	
11		+		+	−		−	
12		+		+	−		−	
14				−				
16				−	+			
17			−		+			
18	−	−				+		
19			−					
20	−	−					+	
21			−	−		+	+	
22			−	−		+	+	
23			+	+		−	−	
24			+	+		−	−	
31				+	−			
32				+		−		
35			+		−			
36			+				−	
37		+						
38		+				−	−	−
39	+				−			
40	+							
41				−	+	+	+	+
42			−		+	+	+	+
43		−			+	+	+	+
44	−				+	+	+	+
Comparator Output	1	1	1	1	0	0	0	0

+ represents a transistor short circuit.
− represents a transistor open circuit.

for this reason that the α-β-γ switch is provided for comparator cutout in Fig. 2. It is assumed that this switch will be thrown shortly after a permanent failure in A, B, or C, as indicated by the alarm lights at g. The possibility of a second failure between failure and cutout is ignored.

Failure-Law Assumptions

Considering permanent failures only, we may assume the reliability of a circuit, *i.e.*, probability of no permanent failure to time t, to be of the usual exponential form $e^{-\xi t}$ where ξ is the *failure rate*. It is assumed that the circuit is transistorized and that ξ is proportional to the complexity as measured by the transistor count n. Thus,

$$\xi = n\psi$$

where ψ is called the failure rate of a transistor.

It is assumed that intermittent errors have a Poisson distribution, which is to say that the probability that a circuit will err on a given cycle is independent of what has occurred in preceding cycles. This may be considered unrealistic in that serial correlation is likely to be present. However, it has been shown by Einhorn and Thiess[4] that under fairly general conditions this assumption leads to a pessimistic estimate of reliability as opposed to the assumption that the sequence of error probabilities form a Markov chain. This does not necessarily yield a pessimistic estimate of gain. The foregoing statement applies to our estimates of the reliability of both the triplicated and untriplicated circuits. Some apparently difficult mathematical problems lie in this area, and there is a dearth of empirical data as to rates and distributions of intermittent errors.

Under the independence assumption, the probability of not exceeding k errors in time t is

$$\sum_{j=0}^{k} e^{-\zeta t}(\zeta t)^j/j!$$

where ζ is the intermittent error rate. Like ξ, ζ is assumed to be proportional to the transistor count (although it is not expected that a transistor itself will become intermittent):

$$\zeta = n\phi,$$

where ϕ is called the error rate per transistor.

Derivation of Reliability Formula

The reliability $R(t)$ of the triplicated system is defined as the probability that no more than k errors will be produced in the system output in time t. This could be caused by any of the following:

1) Permanent failure of one of the input circuits A, B, or C, followed by failure of the input chosen for operation after comparator cutout.
2) Permanent failure in the comparator before cutout. (It is assumed pessimistically that intermittence in the inputs A, B, C is a sufficiently great problem that permanent failures according to the entries in Columns II to VI of Table I are counted, in addition to those of Columns I and VIII, as permanent failures of the system.)
3) Exceeding k intermittent errors in time t.

The following rates are defined:

β = the average rate of intermittent errors in the output of the triplicated system,
μ = the rate of permanent failures in the comparator,
λ = the permanent failure rate of one input (A, B, or C),
ρ = the intermittent error rate of one input (A, B, or C).

It is assumed that intermittent errors do not occur at two different places in the system on the same computational cycle. This means that β is derived only from Columns I and VIII of Table I.

[4] S. J. Einhorn and F. B. Thiess, "Intermittence as a stochastic process," *Proc. NYU-RCA Working Conference on Theory of Reliability*, Ardsley-on-Hudson, N. Y.; April 17–19, 1957.

The reliability of the system will be expressed as the sum of two probabilities,

$$R(t) = A(t) + B(t),$$

where

$A(t)$ = the probability of no cutout, no comparator failure, and k or less errors in time t,

and

$B(t)$ = the probability of exactly one input failure, no comparator failure before cutout, and k or less errors in time t.

The expression for $A(t)$ can be written directly as the product of three independent probabilities:

$$A(t) = e^{-3\lambda t}e^{-\mu t}\sum_{j=0}^{k} e^{-\beta t}(\beta t)^j/j!.$$

The expression for $B(t)$ is more complex, and its evaluation involves approximations. Let

$f(t)$ = the probability of exactly one input failure in time t and no comparator failure before cutout.

Then

$$f(t) = \int_0^t e^{-(3\lambda+\mu)\tau}e^{-\lambda(t-\tau)}3\lambda d\tau$$
$$= \frac{3\lambda}{2\lambda+\mu}e^{-\lambda t}(1 - e^{-(2\lambda+\mu)t}).$$

Here, the variable of integration is the time τ to cutout. Let

$g(t)$ = the probability of exactly one input failure in time t with no comparator failure before cutout and more than k intermittent errors.

Then

$$B(t) = f(t) - g(t)$$
$$R(t) = A(t) + f(t) - g(t).$$

It remains only to evaluate $g(t)$.

Considering m errors occurring before cutout and n errors after cutout, we have

$$g(t) = \sum_{j=k+1}^{\infty}\sum_{m=0}^{j} h_{m(j-m)}$$

where, letting $\delta = \rho - 2\lambda - \mu - \beta$,

$$h_{mn} = \int_0^t e^{-(3\lambda+\mu)\tau}\frac{(\beta\tau)^m}{m!}e^{-\beta\tau}\frac{\rho^n(t-\tau)^n}{n!}e^{-\rho(t-\tau)}e^{-\lambda(t-\tau)}3\lambda d\tau$$
$$= 3\lambda e^{-(\rho+\lambda)t}\frac{t^{m+n}\beta^m\rho^n}{m!n!}\int_0^t e^{\delta\tau}\left(\frac{\tau}{t}\right)^m\left(1-\frac{\tau}{t}\right)^n d\tau.$$

As before, the variable of integration is time to cutout.

Should it be the case that $\delta \leq 0$, then an upper bound on $g(t)$ [which leads to a pessimistic estimate of $R(t)$] could be computed with $\delta = 0$, whereupon geometric summation could be used to transform the formula for $g(t)$ into a multiple of the Poisson series which is tabulated, *e.g.*, by Molina.[5]

However, ordinarily we will have $\delta > 0$, and this will be assumed henceforth. Two approximation methods are given to compute $g(t)$. The first method includes an assumption (that β is negligible) which is optimistic, but ordinarily very slightly so. The second method is more complicated, but is strictly pessimistic.

First Method: Here it will be assumed that β is negligible compared to ρ (which amounts to assuming that the transistor count of input circuits A, B, or C is large compared to six). By the same token, it is assumed that intermittent errors before cutout are negligible. We then have

$$g(t) \approx \sum_{n=k+1}^{\infty} h_{0n}$$
$$= \sum_{n=k+1}^{\infty} 3\lambda e^{-(\rho+\lambda)t}\frac{\rho^n}{n!}\int_0^t e^{\delta\tau}(t-\tau)^n d\tau.$$

Since

$$\frac{1}{n!}\int_0^t e^{\delta\tau}(t-\tau)^n d\tau = \frac{e^{\delta t}}{\delta^{n+1}}\left\{1 - \sum_{j=0}^{n}\frac{(\delta t)^j}{j!}e^{-\delta t}\right\},$$

we have

$$g(t) \approx \frac{3\lambda}{\rho}e^{-(\lambda+\rho)t}\sum_{n=k+1}^{\infty}\sum_{j=n+1}^{\infty}\left(\frac{\rho}{\delta}\right)^{n+1}\frac{(\delta t)^j}{j!}$$
$$= \frac{3\lambda}{\rho}e^{-(\lambda+\rho)t}\sum_{j=k+2}^{\infty}\frac{(\delta t)^j}{j!}\sum_{i=k+2}^{j}\left(\frac{\rho}{\delta}\right)^i$$
$$= \frac{3\lambda}{2\lambda+\mu}e^{-(\lambda+\rho)t}\left\{\sum_{j=k+2}^{\infty}\frac{(\rho t)^j}{j!} - \left(\frac{\rho}{\delta}\right)^{k+1}\sum_{j=k+2}^{\infty}\frac{(\delta t)^j}{j!}\right\}.$$

This last formula is readily computed using Poisson tables.[5]

Second Method: We may rewrite the formula for $g(t)$ as follows:

$$g(t) = 3\lambda e^{-(\rho+\lambda)t}\sum_{j=k+1}^{\infty}\sum_{m=0}^{j}\frac{(\beta/\rho)^m(\rho t)^{j+1}}{\rho(j+1)!}\Phi(m+1, j+2; \delta t)$$

where Φ is defined by the formula

$$\Phi(m+1, j+2; \delta t) = \frac{(j+1)!\int_0^1 e^{\delta ut}u^m(1-u)^{j-m}du}{m!(j-m)!}.$$

Regarding Φ as a weighted averaging of $e^{\delta t}$, it can be shown that $\Phi(m+1, j+2; \delta t)$ decreases as j increases and increases as m increases. Therefore

[5] E. C. Molina, "Poisson's Exponential Binomial Limit," D. Van Nostrand Co., Inc., New York, N. Y.; 1942.

$$g(t) \leqq 3\lambda e^{-(\rho+\lambda)t} \sum_{j=k+1}^{\infty} \sum_{m=0}^{j} \frac{(\beta/\rho)^m (\rho t)^{j+1}}{\rho(j+1)!} \Phi(m+1, k+3; \delta t)$$

$$\leqq \frac{3\lambda}{\rho} e^{-(\rho+\lambda)t} \left(\sum_{j=k+1}^{\infty} \frac{(\rho t)^{j+1} - (\beta t)^{j+1}}{(j+1)!} \right) \cdot \left(\sum_{m=0}^{\infty} (\beta/\rho)^m \Phi(m+1, k+3; \delta t) \right) \cdot$$

This formula may be used to compute an upper bound on $g(t)$. Poisson tables[5] provide the first sum. Since $\Phi(m+1, k+3; \delta t)$ is bounded above by $e^{\delta t}$, the second series will converge rapidly for small β/ρ. Below are recurrence formulas for Φ, derived from the fact that Φ is a confluent hypergeometric function:[6]

$$\Phi(1, k+3; \delta t) = \frac{(k+2)!}{(\delta t)^{k+2}} \sum_{i=k+2}^{\infty} \frac{(\delta t)^i}{i!},$$

$$\Phi(2, k+3; \delta t) = \frac{(k+2)!}{(\delta t)^{k+1}} \sum_{i=k+1}^{\infty} \frac{(\delta t)^i}{i!} - (k+1)\Phi(1, k+3; \delta t),$$

$$\Phi(m+1, k+3; \delta t) = \frac{1}{m} \{(k+3-m)\Phi(m-1, k+3; \delta t) + (2m-3-k+\delta t)\Phi(m, k+3; \delta t)\} \quad \text{for } m \geq 2.$$

Example

As an illustrative example, let us consider the triplication of a computer containing 10^4 transistors. Assume that the permanent failure rate per transistor is $\psi = 10^{-5}$ failures per hour. This is roughly 10 times the transistor failure rates presently being experienced; however, it makes allowance for other parts, and is consistent with being in the position of having less reliability than predicted and therefore being in need of redundancy.

Assume that the required time of operation is one hour. Then $\lambda = 10^4\psi = 0.1$, $t = 1$, and the reliability of the computer as regards permanent failures is $e^{-0.1} = 0.9$.

Assume that the allowable number k of errors in time t is 10 and that, by this standard, intermittence causes half as much unreliability as permanent failures, *i.e.*,

$$0.95 = \sum_{j=0}^{10} e^{-\rho t}(\rho t)^j / j!,$$

which corresponds to $\rho = 6.2$ errors per hour, and $\phi = 6.2 \times 10^{-4}$ errors per hour, the intermittent error rate per transistor.

As noted earlier, β is derived from Columns I and VIII of Table I, and μ is derived from all eight columns. We have:

$$\beta = 6\phi = 0.0037 \text{ error per hour}$$

$$\mu = 30\psi = 0.0003 \text{ failure per hour.}$$

[6] A. Erdélyi, W. Magnus, F. Oberhettinger, and G. Tricomi, "Higher Transcendental Functions," Bateman Manuscript Project, McGraw-Hill Book Co., Inc., New York, N. Y.; 1953.

Clearly β is negligible compared to ρ. Using these values and the first of the two methods above, we compute the reliability with triplication to be $R(t) \approx 0.985$. The reliability without triplication is $0.9 \times 0.95 = 0.855$. Thus, the gain accrued by triplication is approximately $(1-0.855)/(1-0.985) = 9.7$.

If this example is recomputed with an error rate corresponding to an intermittence reliability of 0.975 instead of 0.95, then $R(t)$ remains virtually unchanged. The same is true if k is changed from 10 to 100 in both instances.

In the above example, the estimated values of $g(t)$ were computed to be 0.0018 and 0.0022 by the first and second methods respectively. In the alternative three cases of the preceding paragraph, $g(t)$ and, accordingly, the difference between the results of the two methods are much smaller. This confirms that neglecting β in computing $g(t)$ does not have a significant effect in these cases.

The gain factor of 9.7 is impressive. The gain computed with the same failure and error rates per transistor, but with the assumption that the computer is triplicated has only 1000 or 100 transistors, becomes 25 or 3.3 respectively.

Suppose the computer with 10^4 transistors consists of a series of 10 stages with 1000 transistors each. With triplication, one might take a vote at each stage. The problem of estimating the gain of such a system is complicated by the numerous states (and corresponding intermittent error rates) which can occur as one or more cutouts take place. However, if one assumes that at most one cutout will occur in time t, the situation being analogous to the single-comparator case. The formulas already derived may be adapted by substituting 10β, 10μ for β, μ and making other substitutions. Analysis proceeding in this fashion could be used to determine the optimum level of vote-taking.

Discussion of the Evaluation

The natural simplified method of evaluating a voting comparator is to ignore intermittence and comparator reliability. Then the gain factor without cutout is approximately $\frac{1}{3}q$ where q is the failure probability of the circuit being triplicated. The results computed from the formula for $R(t)$ derived above are quite different from this simplified method.

It is not desired to generalize too far on such comparisons, since the present formula is conditioned by various uncertainties, notably the form of the distribution of intermittent errors. It is possible that our treatment of comparator failures is too pessimistic; however, this could easily be modified as empirical information may warrant.

It is believed thai the methods developed here can be used to provide a basis for deciding whether or not to triplicate and the optimum level at which the vote should be taken. While much further work needs to be done, particularly in the area of intermittence, such estimates should be somewhat closer to reality than would be obtained, for example, by ignoring intermittence or comparator reliability or both.

ACKNOWLEDGMENT

The research reported here was carried out under USAF Contract No. AF 04(645)-23, and much of it was performed by associates of the authors. Figs. 2 and 3 were conceived by J. K. Moore and S. Schneider, Fig. 4 by D. J. Kerrisk and G. H. Barnes, and Fig. 1 by J. C. Moffet. The comparator evaluation was carried out largely by S. J. Einhorn, G. S. Goodman, and F. B. Thiess, the second of the two methods for computing $g(t)$ being due to the latter. H. H. Goode, of the University of Michigan, proposed the idea of forced-failure checking which was investigated, along with low-level redundancy techniques, by D. J. Kerrisk. Thanks are due to Mrs. A. S. Biklé for editorial assistance.

Discussion

O. Lowenschluss (Sperry Gyroscope): It is possible to build a 6-transistor DCTL majority element, outside of checking features, so why use the 53 transistors?

Mr. Wagner: The case that we were just discussing was done in this way because it represented more general cases and because it provides for indication outputs, and one can monitor the operation, check and determine the volts during the operation.

Mr. Lowenschluss: Also, of course, the checking features are worth something.

R. M. Walker (IBM): Could you give a comparison (for triplication) in terms of the ratio of improvement in mean time to failure?

Mr. Wagner: When you are at the upper end of the probability scale the 90–100 per cent failure probability is approximately inversely proportional to mean time so that the gain in the mean time would be again about a factor of ten in the example cited. This assumes that the process is described by exponential failure law, and regarding the triplicated system, as a whole, I doubt if this is strictly the case, especially with two different states to consider, before and after cut-out. But for a rough answer, the ratio of mean time would be approximately the same as the ratio between the failure probabilities

Analog Logarithmic and Antilogarithmic Circuits Using Switching Transistors

A. J. SCHIEWE[†] AND K. CHEN[‡]

INTRODUCTION

AS THE FUNCTION of modern control and computer systems becomes more complex, there is an increasing need of electronic analog circuits which automatically perform the mathematical operations of taking the logarithm and antilogarithm. The logarithmic function may be desired in itself as in conversion of linear inputs into logarithmic values for scale compression in one-dimensional instrument indicators and recorders. On the other hand, it may be used in conjunction with the antilogarithmic function to obtain the operations of multiplication, division, and taking powers and roots.

The circuits discussed in this paper were developed with the following objectives:

1) Reliability and indefinitely long life.
2) Reproducible characteristics.
3) Fast response (less than 1 millisecond).
4) Good accuracy (less than 1 per cent error).
5) Simplicity and practical design.
6) Moderate range of inputs (2 decades).
7) Temperature stability (-55°C to $+71$°C).

[†] The Ramo-Wooldridge Corp., Los Angeles, Calif.
[‡] Westinghouse Electric Corp., East Pittsburgh, Pa.

To the authors' knowledge no contemporary circuits have satisfactorily met all these objectives.

The computing elements described herein are completely static in their operation utilizing exponential time decays in conjunction with pulse-width modulation. The output is in the form of a train of voltage pulses, the average value of which is a definite function of the input voltage.

BASIC COMPUTING ELEMENT

Fig. 1 shows the basic computing circuit used to generate the logarithmic and antilogarithmic functions. That it may also be used to generate other transcendental functions will be demonstrated. The *n-p-n* transistor will block (exhibit a very high impedance between emitter and collector) if the base is more negative than both the emitter and collector. If the base is more positive than the emitter, the transistor will exhibit nearly a short circuit between emitter and collector within the limit of its base-to-collector current gain.

In order to generate the logarithmic function, a dc voltage, $e_1 = E_1$, is used to modulate the zero level of a train of exponential pulses e_2 [Fig. 2(a)]. The relative polarities are as indicated in Fig. 1. When the net voltage, e_b, is negative, the transistor blocks and the voltage,

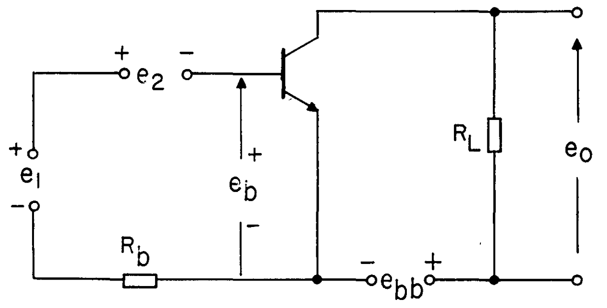


Fig. 1—Basic computing element.

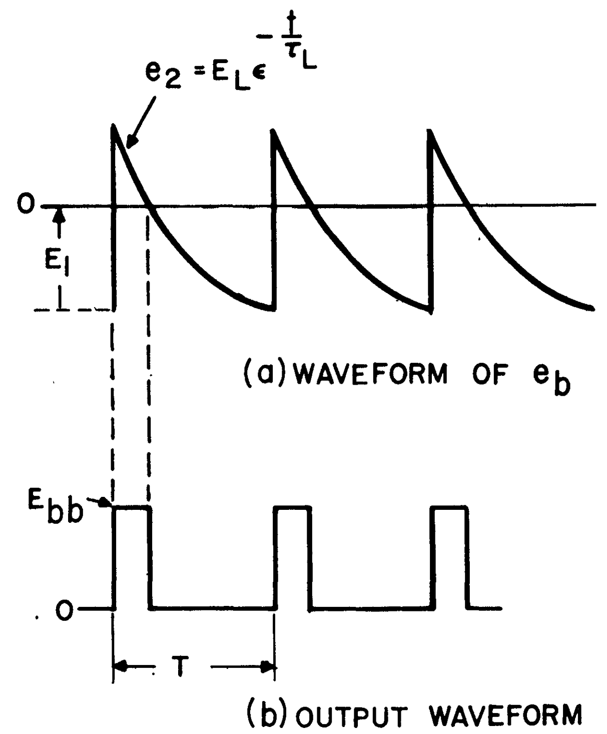


Fig. 2—Waveforms pertinent to logarithmic circuit.

e_0, is zero. When, on the other hand, the net voltage is positive, the transistor conducts applying a dc voltage, $e_{bb} = E_{bb}$, across the output. The resultant output voltage is a train of rectangular pulses of amplitude E_{bb} and with a width W_L. That the width, W_L, is a logarithmic function of the input, E_1, is easily seen in the following manner. Over the period T the voltage e_2 is given by

$$e_2 = E_L \exp - t/\tau_L. \tag{1}$$

The transistor switches from conduction to blocking when E_1 equals e_2 in magnitude. Hence, the conduction time and consequently the width, W_L, of the output pulses is given by

$$W_L = -\tau_L \ln E_1/E_L. \tag{2}$$

If only the logarithmic function is desired, the output voltage may be averaged to obtain a dc voltage, E_0, proportional to the logarithm of the input E_1

$$E_0 \text{ (average)} = E_{bb}W_L/T = -E_{bb}\tau_L f \ln E_1/E_L. \tag{3}$$

where f is the frequency of the exponential pulse train.

The inverse function, or antilogarithmic function, is obtained in the following manner. It is assumed that the antilogarithmic circuit is to be operated in conjunction with the logarithmic circuit already described. That is, the input information to the antilogarithmic circuit will be contained in a pulse width, W_A, which results from mathematical operations being performed on the outputs of the logarithmic circuits. In this case the input, e_1, becomes the rectangular pulse train of Fig. 3(a) and the voltage e_2 becomes a dc bias voltage to insure proper switching of the transistor. As one might expect, the voltage, e_{bb}, is now required to be a train of exponential pulses synchronized with the input pulses e_1 as indicated in Fig. 3. The exponential pulses are switched to the

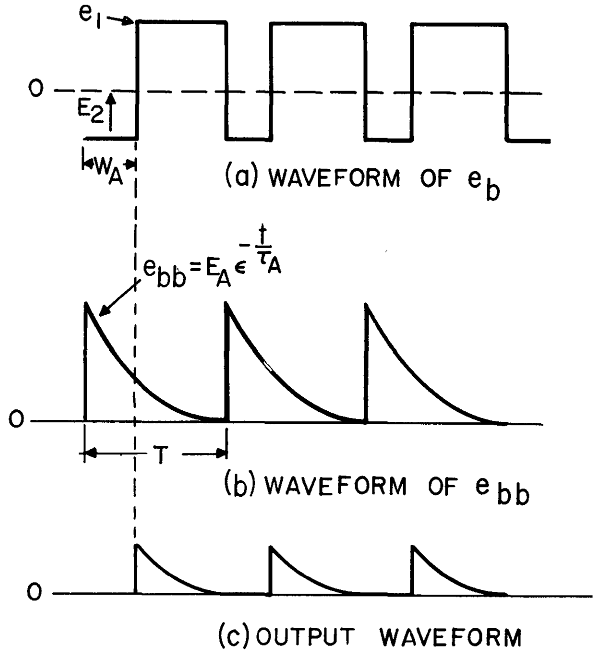


Fig. 3—Waveforms pertinent to antilogarithmic circuit.

output only during the interval $(T - W_A)$ in each period T. The average output voltage, E_0 is given approximately by

$$E_0 \text{ (average)} \cong \frac{E_A}{T} \int_{W_A}^{\infty} \epsilon^{-t/\tau_A} dt \tag{4}$$

$$= E_A \tau_A f \epsilon^{-W_A/\tau_A}$$

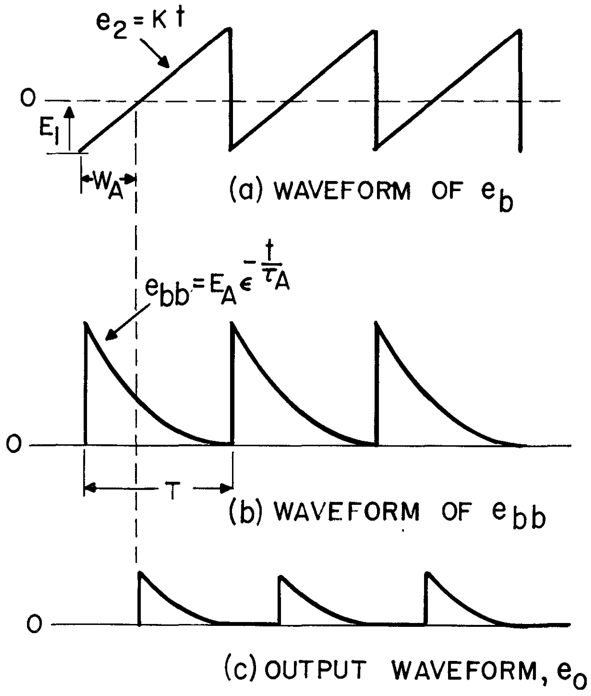
$$= E_A \tau_A f \text{ antiln } (-W_A/\tau_A). \tag{5}$$

A constant error is involved in this approximation in that a portion of the exponential tail has been deleted in the actual averaging process. The magnitude of this error relative to full-scale output may be made very small by choosing the time constant τ_A small. Also, since the error is constant and independent of the input variable, W_A, it may be removed completely by adding a bias voltage at the output. The magnitude of the error is given by

$$\text{error} = E_A \tau_A f \epsilon^{-T/\tau_A}. \tag{6}$$

If for some reason the input to the antilogarithmic circuit is scaled into a dc voltage, E_1, then it is necessary to make e_2 a sawtooth waveform as indicated in Fig. 4. The resultant output waveform and hence average output voltage is the same as that described in the previous paragraph with W_A now proportional to E_1.

$$\langle E_0 \rangle_{Av} = E_A \tau_A f \text{ antiln } (-KE_1/\tau_A). \tag{7}$$



$e_2 = Kt$
0
E_1
W_A
(a) WAVEFORM OF e_b
$e_{bb} = E_A \epsilon^{-\frac{t}{\tau_A}}$
0
T
(b) WAVEFORM OF e_{bb}
0
(c) OUTPUT WAVEFORM, e_0

Fig. 4—Waveforms for an alternate antilogarithmic circuit.

Indeed, a great variety of transcendental functions of an input voltage can be obtained in theory from this very simple circuit. Table I is a listing of a few such

TABLE I
AVERAGE OUTPUT VOLTAGES FOR VARIOUS WAVEFORMS OF e_2 AND e_3

e_1	e_{bb}	e_2	E_0 (average) Proportional To:
dc	dc	Kt	$e_{bb} \cdot e_1$
dc	dc	Kt^2	$e_{bb} \cdot e_1^{\frac{1}{2}}$
dc	dc	Kt^n	$e_{bb} \cdot e_1^{1/n}$
dc	dc	$K \cos \omega t$	$e_{bb} \cdot \cos^{-1} e_1/K$
dc	dc	$K \sin \omega t$	$e_{bb} \cdot \sin^{-1} e_1/K$
dc	dc	$K\epsilon^{-\alpha_t}$	$e_{bb} \cdot \ln e_1/K$
dc	$\cos \omega t$	Kt	$\sin \omega/Ke_1$
dc	$K_1 t^n$	Kt	e_1^{n+1}
dc	$f(t)$	Kt	$\int^{e_1/K} f(t)dt$
dc	$K_1 t^n$	Kt^m	$e_1^{(n+1)/m}$
dc	$\cos \omega t$	Kt^2	$\sin (\omega/\sqrt{K})\sqrt{e_1}$

functions. The input, e_1, is assumed present as a dc voltage. The voltages e_2 and e_3 are assumed periodic with period T. The average output voltage is found in each case from the same line of reasoning as given in the preceding paragraphs. The authors have developed detailed circuitry only for the logarithmic and antilogarithmic functions since only the easily obtained exponential is required for e_2 and e_3. Van Allen and Schaefer used somewhat the same principle to obtain transcendental functions, but they required transistors and magnetic cores.[1] Glaser and Blasbalg developed a logarithmic voltage quantizer with vacuum tube circuitry based on the principle of modulating exponential voltage pulses, but they did not consider the antilogarithmic counterpart, nor transistor circuitry in performing the logarithmic function.[2]

OPERATIONS ON PULSE WIDTHS

It is possible, of course, to average the output of the logarithmic circuit with an approximate filter and then combine several such dc outputs to obtain a dc input for the antilogarithmic circuit. For multiplication the dc outputs from the logarithmic circuits would add and for division, subtract. To take powers and roots, the dc output of a single logarithmic circuit would be proportioned. However, the filtering of the logarithmic outputs greatly increases the response time of the over-all computer operation. For this reason a magnetic core is utilized to operate on the pulse widths directly.

If such a core is assumed to be at positive saturation and provided with low-resistance windings of N_i turns, then a set of rectangular nonoverlapping voltage pulses of proper polarity from low internal impedance sources across the windings will tend to reset the core in accordance with (8), provided negative saturation is not reached (Fig. 5).

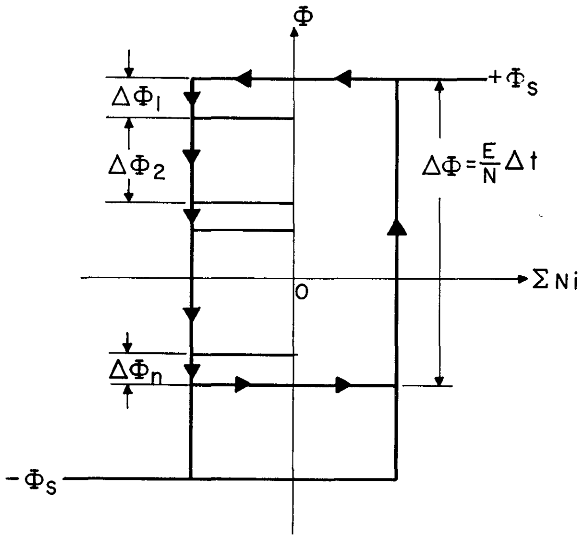


Fig. 5—Magnetic characteristic of rectangular hysteresis loop core material.

$$\Delta\phi = \sum_{i=1}^{n} \Delta\phi_i = \sum_{i=1}^{n} E_{bbi}\Delta t_i/N_i. \tag{8}$$

[1] D. H. Schaefer and R. L. Van Allen, "Transcendental function analogue computation with magnetic cores," *AIEE Trans.*, vol. 75, pp. 160–165; May, 1956.

[2] E. M. Glaser and H. Blasbalg, "A logarithmic voltage quantizer," IRE TRANS., vol. EC-4, pp. 15–155; December, 1955.

The Δt_i is the time duration of the ith pulse and E_{bbi} is its amplitude. In order to bring the core back to positive saturation, an equal and opposite change in flux must be effected. If a constant "firing" voltage E is applied to a winding of N turns, then the time duration of this voltage must produce the same volt-time area per turn as on reset to saturate the core.

$$E\Delta t/N = \sum_{i=1}^{n} E_{bbi}\Delta t_i/N_i. \quad (9)$$

All the Δt_i's are simply the widths W_i of rectangular pulses and Δt is the width W of the "firing" voltage pulse.

$$W = N/E \sum_{i=1}^{n} E_{bbi}W_i/N_i. \quad (10)$$

The above discussion applies equally well if the operation is made cyclic in nature. The W_i's represent the outputs of identical logarithmic circuits each of which has been assigned a discrete interval of time, $T/2n$ seconds, for full range of operation (where $\frac{1}{2}T$ is the resetting period and n is the number of log circuits being used). The pulses from the logarithmic circuits are made nonoverlapping by means of time delays to be described later. During the "read out" half cycle the logarithmic circuits are disconnected from the core and the firing voltage E is applied. The width of the voltage pulse across the core is given by (10) and depicted in Fig. 6. Substituting (2) into (10) yields

$$W = -N/E \sum_{i=1}^{n} E_{bbi}\tau_L/N_i \ln E_{1i}/E_{Li}. \quad (11)$$

If this W is made the input to the antilogarithmic circuit previously described, (5) may be applied resulting in

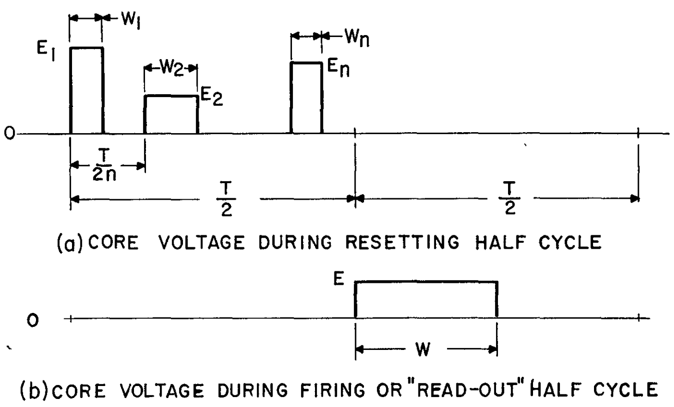
$$E_0 \text{ (average)} = E_A\tau_A f\left[\prod_{i=1}^{n}\left(\frac{E_{1i}}{E_{Li}}\right)\left(\frac{N}{N_i}\frac{E_{bbi}}{E}\frac{\tau_L}{\tau_A}\right)\right] \quad (12)$$

From (12) it is clear that logarithmic-magnetic core-antilogarithmic ensemble may be used to obtain multiplication and the taking of powers and roots. If division is to be performed, then the divisor must be presented to the core as a negative pulse and must be scaled and delayed in such a fashion that it does not return the core to positive saturation before the "read out" half cycle.

The delay time for the ensemble is one cycle of the operating frequency, f. If the output must be filtered before use, then the filtering delay must be included. For the circuits developed a frequency of 1000 cps was used giving a delay of one millisecond plus filtering delay.

Logarithmic Circuit

Details of the logarithmic circuit, which includes the network for generating exponential pulses, are shown in Fig. 7. The main difference between the logarithmic



W_1
E_1
W_2
E_2
W_n
E_n
0
$\frac{T}{2n}$
$\frac{T}{2}$
$\frac{T}{2}$
(a) CORE VOLTAGE DURING RESETTING HALF CYCLE
E
0
W
(b) CORE VOLTAGE DURING FIRING OR "READ-OUT" HALF CYCLE

Fig. 6—Pulse-width operations.

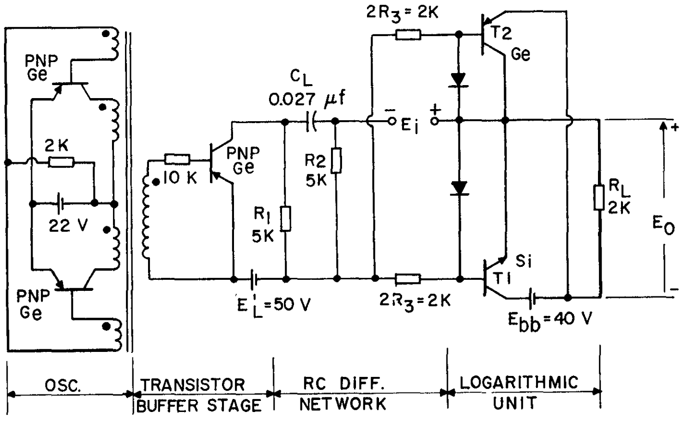


Fig. 7—Details of the logarithmic circuit.

unit in this circuit and the basic circuit shown in Fig. 1 is the inclusion of a second transistor switch T2 of the germanium p-n-p type for the purpose of minimizing the effect of leakage current through the main transistor switch T1. Since this second switch T2 is closed whenever the main switch T1 is opened and since T2 is connected in the inverted connection,[3] its extremely low voltage drop effectively shunts the leakage current of T1.

The exponential pulses required for the logarithmic operation are obtained from a dc source by means of a magnetic-coupled transistor square-wave generator,[4] a transistor buffer stage and a RC differentiating network as shown in Fig. 7. The diodes connected to the bases of T1 and T2 maintain the input impedance seen by the exponential function generator essentially constant over a full cycle. The diodes also serve the purpose of reducing the Zener voltage requirement of T1 and T2.

The input voltage E_i is so connected that it modulates the zero level of the base current of T1 during the half cycle in which the capacitor C_L discharges. An analysis of the circuit transient shows that the time constant as-

[3] R. L. Bright, "Junction transistors used as switches," *AIEE Trans.*, vol. 74, p. 111; March, 1955.

[4] G. H. Royer, "A switching transistor d-c to a-c converter having an output frequency proportional to the d-c input voltage," *AIEE Trans.*, vol. 74, p. 322; July, 1955.

sociated with this discharge, which becomes the τ_L in (1)–(3) of previous discussion, is given by

$$\tau_L = C_L[R_1 + R_2R_3/(R_2 + R_3)]. \tag{13}$$

The base current of the transistor T1 has the waveform shown in Fig. 8, in which the portions a and b are determined by the relation

$$\frac{b}{a+b} = \frac{R_1R_2 + R_2R_3 + R_3R_1}{R_2(R_2 + R_3)} \cdot \frac{E_i}{E_L'}. \tag{14}$$

Thus, the maximum value for E_i, or the upper limit of the input to the logarithmic circuit, can be determined by setting $a = 0$ and becomes

$$E_{i\,\max} = \frac{R_2(R_2 + R_3)}{R_1R_2 + R_2R_3 + R_3R_1} E'_L. \tag{15}$$

The lower limit of the input $E_{i\ \min}$ is dictated by the minimum base-to-emitter voltage required for the silicon transistor T1 to perform as a closed switch. The choice of the supply voltage levels and the RC component values as given in Fig. 7 was based on the consideration of the Zener voltage and current gain capabilities of the available transistors, and the objective of obtaining maximum operating range of the logarithmic circuit for a supply frequency of 1000 cps.

The steady-state temperature characteristic of the logarithmic circuit is shown in Fig. 9. All transistors and diodes used in the logarithmic circuit under test and in the antilogarithmic circuit to be described are of the following types:

Si diodes	—1N137A
Si *n-p-n* transistors	—904 (Texas Instruments, Inc.)
Ge *p-n-p* transistors	—2N74
Ge *n-p-n* transistors	—4815C.

The only parts of the circuit which were subjected to temperature changes were the semiconductor components in the logarithmic unit. Under these conditions the experimental tests indicate the maximum error to be within 1.5 per cent of the full-scale output for the temperature range from 0°C to +71°C with a corresponding input range of 0.7 to 120 volts.[5] The major cause of error is the lack of sharpness at the trailing edge of the output voltage pulse as the transition time of the transistor switch from conducting to cutoff lengthens at low input. The maximum error increased above the value of 1.5 per cent as temperature decreased from 0°C to −60°C. This effect is believed to have been caused by a significant decrease in current gain of the silicon transistor for temperatures in this range.

The stable operation of the logarithmic circuit within the temperature range of 0°C to +71°C is attributed to the operation of the transistors in the switching mode. For the same reason the circuit possesses the important property of reproducibility. The characteristics do not

[5] The errors in measurement were estimated to be 1 per cent.

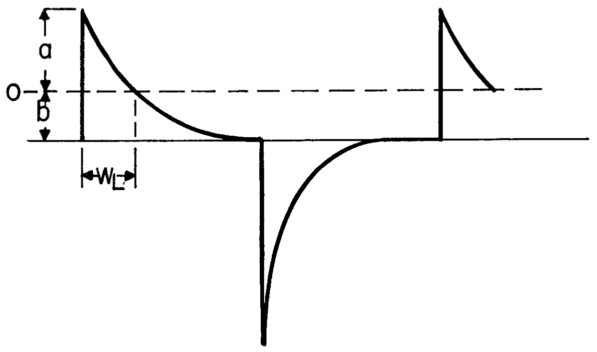


Fig. 8—Base current waveform of the main transistor switch in the logarithmic circuit.

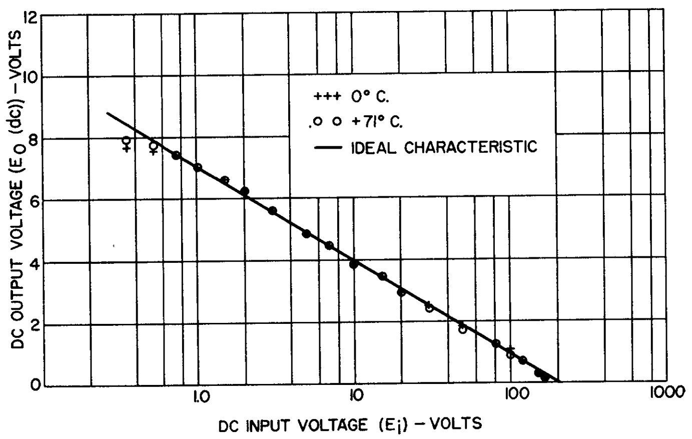


Fig. 9—Steady-state temperature characteristic of the logarithmic circuit.

depend upon a particular transistor or upon special matching of components which is experienced in many contemporary circuits. Also, simplicity has been achieved even if the generation of the exponential pulses is taken into consideration. The circuit gains reliability and long life from the use of transistors and magnetic cores as the only active elements.

Antilogarithmic Circuit

Details of the antilogarithmic circuit are shown in Fig. 10 (next page). Here the input is assumed to be a dc voltage. A sawtooth function generator, which mixes a square wave with a triangular wave in proper proportion to produce the desired output waveform, is included in the circuit for the purpose of converting the dc input E_i to the form of pulse width ($\frac{1}{2}T - W_A$). The exponential voltage pulses required as the voltage supply for the output of the antilogarithmic unit are generated by periodically charging and discharging the 0.04 μf capacitor (lower portion of Fig. 10). The germanium *p-n-p* transistor T2 and the two diodes in the antilogarithmic unit are employed for the same reasons given in the discussion of the logarithmic circuit.

The steady-state characteristic of the antilogarithmic circuit at room temperature is shown in Fig. 11. The maximum error is within 1 per cent of the full-scale output for an output voltage range from 0 to 1.5 volts. The

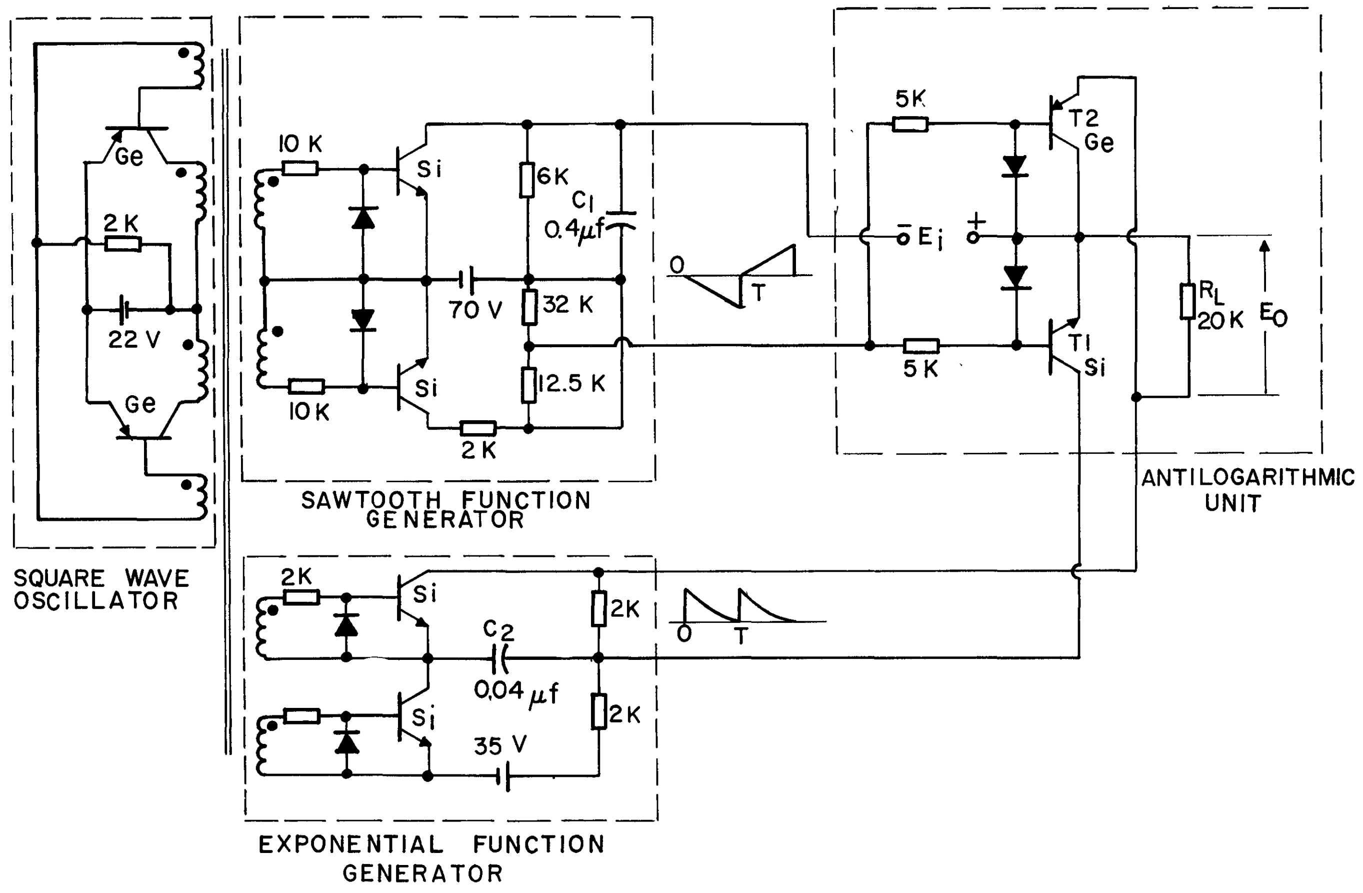


Fig. 10—Details of the antilogarithmic circuit.

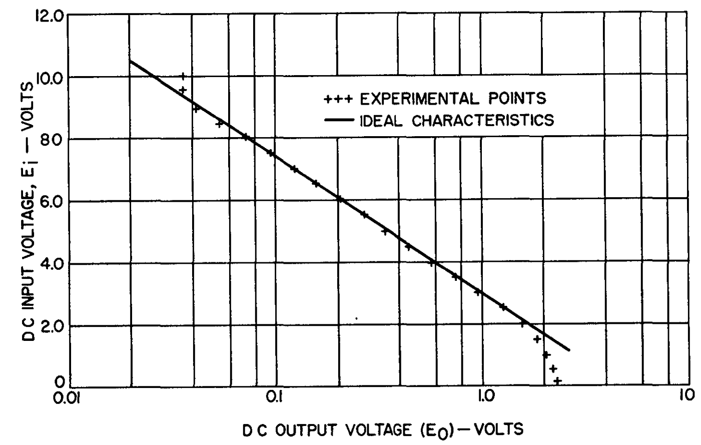


Fig. 11—Steady-state characteristic of the antilogarithmic circuit.

major source of error is the inherent approximation given by (5) and (6) and the imperfect linearity of the sawtooth waveform. Comments on the reproducibility, reliability, and long life of the logarithmic circuit apply also to the antilogarithmic circuit. Simplicity and temperature stability of the antilogarithmic circuit in Fig. 10 are somewhat poorer because of the employment of the low-voltage sawtooth function generator, which, however, is not needed in the ensemble circuit described below.

Logarithmic-Magnetic Core-Antilogarithmic Ensemble

Details of the ensemble circuit with two inputs are in Fig. 12 (opposite). The two logarithmic units, the pulse-width algebraic unit, and the antilogarithmic unit are the heart of the ensemble. Accessory equipment consists of several transistor buffer stages, several exponential function generators, a square-wave oscillator which synchronizes the operation of the different parts of the ensemble, and a delay network which keeps the output pulses of the logarithmic units from overlapping.

The logarithmic units differ slightly in physical layout from that in Fig. 7. First, they are operating during the charging periods of the capacitors C_1 and C_2. This was necessary in order to be able to introduce the delay network and to use silicon transistors in the logarithmic units. (Only *n-p-n* type of silicon transistors was available at the time of the circuit development.) The inversely connected germanium transistor in Fig. 7 is not used here because it would interfere with the operation of the pulse-width algebraic unit.

The delay network is composed of a magnetic core and a biased rectifier. The core is so designed that it is saturated upon absorbing one-half the volt-time area per pulse put out by the square-wave oscillator. Therefore the operation of the upper logarithmic unit lays behind the operation of the lower one by a quarter cycle.

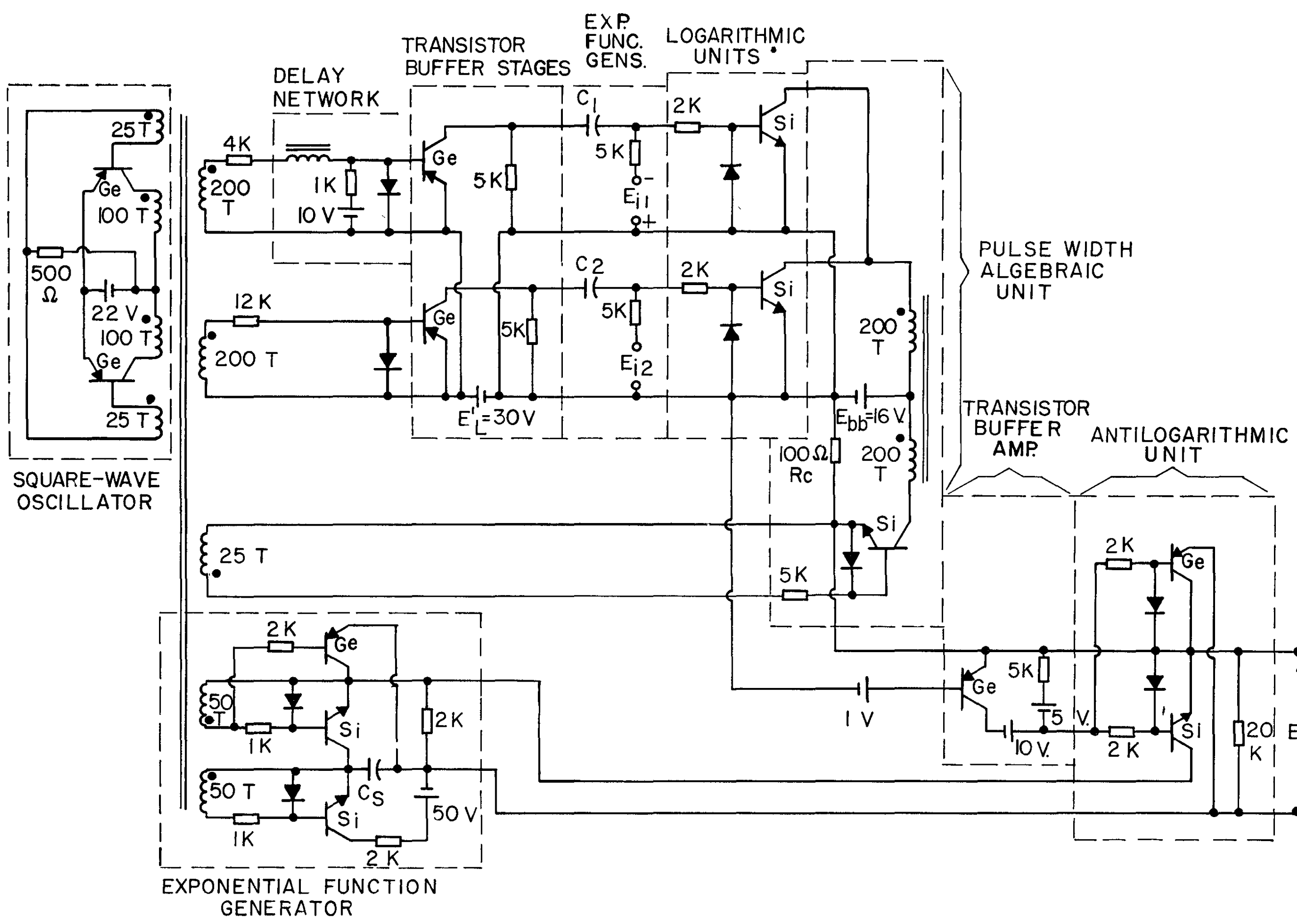


Fig. 12—Ensemble circuit.

The pulse-width algebraic unit operates in accordance with the principle discussed previously. The transistors of the log units close during part of the reset half cycle, and the transistor of the pulse width unit closes during the entire firing half cycle. Since the same voltage E_{bb} is applied to the core during both reset and firing operations, its variation has no first-order effect on the accuracy of the pulse-width algebraic performance.

In order to operate the antilogarithmic unit, a pulse width corresponding to $(\frac{1}{2}T-W_A)$ is required where W_A is the width of the winding voltage pulse of the magnetic core during the firing half cycle. This pulse width of $(\frac{1}{2}T-W_A)$ is easily obtained as the current waveform picked up by the resistor R_c and is then used to drive a transistor amplifier which in turn drives the antilogarithmic unit. All remaining parts of the ensemble circuit have been discussed previously.

The ensemble circuit has been operated as a simple multiplier with two inputs and the resultant characteristic at room temperature is shown in Fig. 13. This was achieved by choosing the circuit parameters so that $\tau_L=\tau_A$, $E_{bb1}=E_{bb2}=E$, $N_1=N_2=N$, and $n=2$ in (12). With the omission of one logarithmic unit and halving the reset voltage (by connecting the emitter of the re-

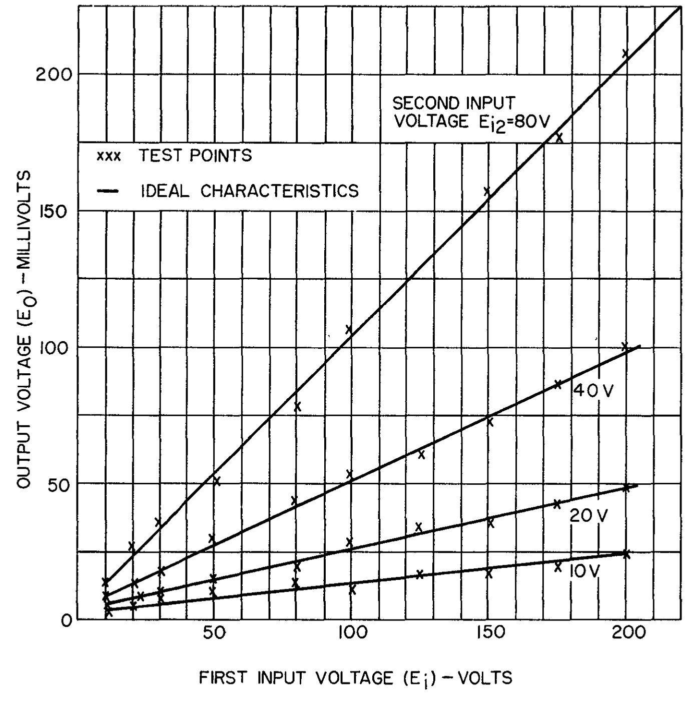


Fig. 13—Steady-state characteristic of the ensemble operating as a multiplier.

maining log unit transistor to the center tap of E_{bb}), the same circuit was made to operate as a square-rooter with the resultant characteristic at room temperature shown in Fig. 14. This operation corresponds to $\tau_L = \tau_A$, $E_{bb1} = \frac{1}{2}E$, $N_1 = N$, and $n = 1$ in (12). Of course, any other operation representable by (12) can be achieved by the highly flexible ensemble circuit in Fig. 12 without any modification of the basic circuit configuration. The requirement for each additional input is a logarithmic unit and a delay network. The over-all accuracy of the ensemble characteristics as indicated by Figs. 13 and 14 is in the order of 2 per cent.

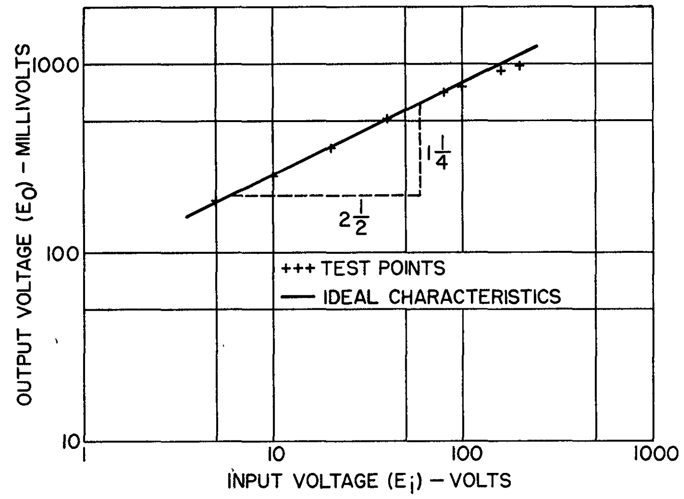


Fig. 14—Steady-state characteristic of the ensemble operating as a square-rooter.

Conclusion

A logarithmic circuit and an antilogarithmic circuit using switching transistors have been developed based on the principle of modulating exponential voltage pulses. The main feature of these circuits is their quality of reliability and reproducibility not found in contempory circuits of the same degree of accuracy. Also, good stability with temperature variations has been achieved by operating the transistors as switches. Time constants of the exponential pulses can be made stable by the use of negative-temperature-coefficient resistors and mica capacitors. The circuit performance also approaches the other objectives listed in the introduction of this paper. An ensemble of the logarithmic and antilogarithmic circuits interconnected through a pulse-width algebraic unit keeps the over-all response time within one cycle of the supply frequency if the output pulse waveform is acceptable. The ensemble has such a high degree of flexibility that it should find applications in many computing systems.

Acknowledgment

The development work which led to this paper was done when both authors were with Westinghouse Electric Corporation. The authors wish to acknowledge the technical guidance rendered them by Dr. G. F. Pittman and R. O. Decker, and the assistance by I. Gerson in the improvement and measurement of the ensemble circuit.

High-Speed Digital-to-Analog Conversion by Integration of a Variable-Rate Pulse Train

A. DEAN GLICK[†]

Introduction

THE CONVERSION of a binary number into an analog voltage, current, or shaft position is a basic problem which must be solved in a wide variety of digital control systems. Because of this widespread need, many techniques have been devised, each with individual limitations and shortcomings. A conversion of digits-to-shaft position can be made by comparing the digital representation from a code wheel with the binary number to be converted. The shaft attached to the code wheel is made to rotate until the reading obtained from it agrees with the number to be converted. The code wheel reading is subtracted from the number to be converted, and the difference quantity generates an analog error voltage which controls a servo connected to the shaft of the code wheel. A digit-to-voltage or current conversion may be accomplished by the method illustrated in Fig. 1. The principle is that of assigning appropriate analog weights to each binary digit. The individual analog quantities are essentially voltages or currents which are proportional to the significance of the respective digits in the numbering system. The con-

[†] Minneapolis-Honeywell Regulator Co., Minneapolis, Minn.

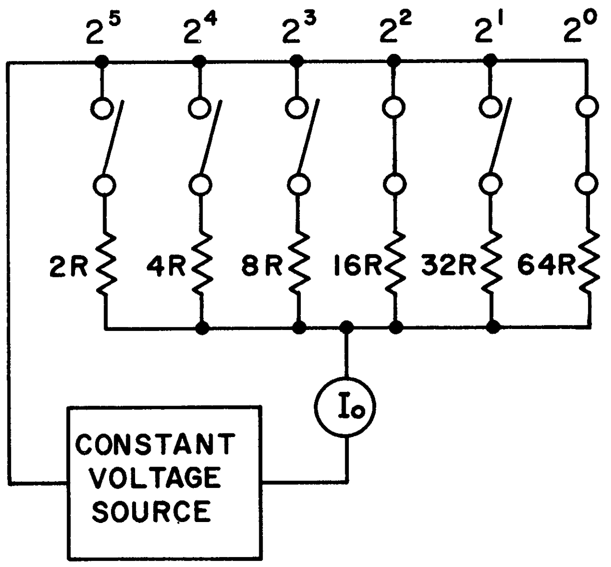


Fig. 1—Digital-analog current converter.

version is effected by summing up the analog equivalent for each digit containing a binary one in the number being converted. In Fig. 1, the conversion is to an analog current, I_0, proportional to the binary number, represented by the switch settings. The binary number represented by the switch setting in this figure corresponds to 000101. It will be noted that the current produced is 5/64 E/R. Another system which should be mentioned, because at first glance it appears similar to the one to be described, is the Shannon-Rack converter. In this system, the binary representation of the number to be converted is sampled serially, starting with the lowest-order digit and proceeding to the highest. As the digits are shifted out of the register containing the number, a one digit will cause a pulse to be generated into a shunt RC circuit. The time constant of the RC network is such that in a digit pulse period the charge on the capacitor will decay to one-half of its initial value. Therefore, after n pulse periods, the charge remaining due to a one occurring at the first pulse time is proportional to $1/2^{n-1}$. The exponential decay provides a convenient method of obtaining the appropriate binary weight. The voltage across the capacitor, one pulse period after the most significant digit has been shifted out, will equal a voltage proportional to the digital number which was contained in the register. A simplified circuit of this system is shown in Fig. 2. The circuit consists of a constant current source, a switch, a resistor, and a capacitor. The switch is controlled by the output of the register containing the number being converted. It is closed for a given period of time as each one is shifted from the register.

In connection with the development of a digital gyro-torquing technique for an airborne digital computer, a novel method for converting a binary number to an analog quantity was developed. Digital gyro torquing is a term used to describe a method of applying varying sequences of constant width, constant amplitude, current pulses to the gyro torque-motor winding. In the analog case, a steady value of current is applied to produce the desired torque. In the digital case, the average rate at which the standardized pulses are applied determines the torque produced. The conversion system developed produces constant charge pulses at a rate proportional to the digital number being converted. Since a charge flow rate is equal to a current, the output is a current proportional to the digital number. This system requires the following:

1) A means of producing an average pulse rate proportional to the digital number to be converted.
2) A means of producing, from this pulse train, current pulses which are precisely controlled in width and amplitude.
3) A means of averaging and smoothing these pulses such that the analog output is proportional to the average current produced by these pulses.

In the subject matter which follows, the system is explained on the basis of converting a normal binary number to a proportional analog current. However, the extension to the conversion to voltage and shaft position is also discussed.

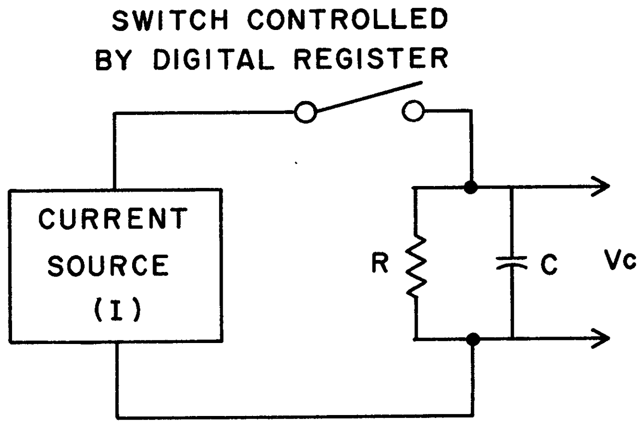


Fig. 2—Shannon-Rack converters.

Generating the Desired Pulse Rate

The method of generating a pulse rate proportional to the digital number to be converted is perhaps the most novel aspect of this system. Ideally, the pulse train generated would contain pulses which were equally spaced with respect to one another. Fig. 3 shows the pulse patterns generated by such an ideal converter. The number of pulses produced in each time period T equals the number being converted. This type of system was abandoned at an early stage because of the difficulty in generating the wide range of pulse rates required, yet maintaining a precise spacing between the pulses. Implementing this type of system required converting the digital number to an analog quantity, which could then be used to control a pulse-generating system in which the pulse rate produced was a function of the analog signal applied. The conversion from digits to analog in this case suffered the usual accuracy limita-

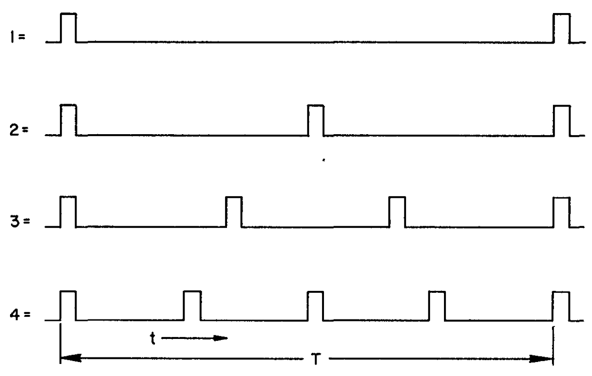


Fig. 3—Ideal pulse patterns.

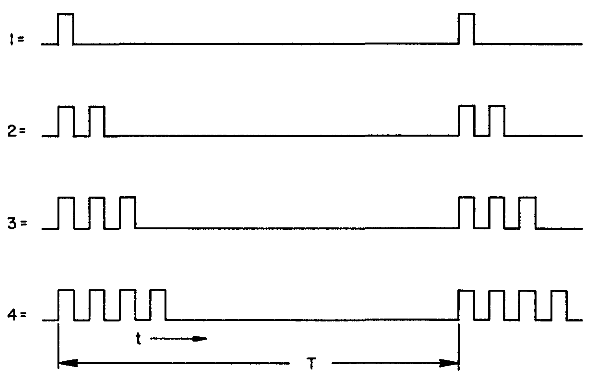


Fig. 4—"Count off" pulse patterns.

tions of such devices and further conversion to a pulse rate was somewhat meaningless.

A more truly digital means of converting a digital number to a pulse rate is to generate pulses as a function of the computer clock (or some other standard) and then gate a number of these pulses into the output line. The number of pulses gated out is made equal to the binary number to be converted. Fig. 4 shows pulse patterns obtained from such a system. As before, the number of pulses produced during each period T equals the binary number being converted. This system has the advantage that discrete positions are allotted for each pulse, and the time at which each pulse occurs is controlled by the computer clock. Since the period T is associated with the same clock source which is causing the pulses to be generated there will always be the precise number of pulses per period T desired. This type of system is easily implemented by having a binary counter which is reset to zero at the beginning of each period T. Pulses are applied to the counter and to the output line at the same time and continue to occur until the counter reaches the digital number being converted. In utilizing this system, the average current produced is a precise conversion of the binary number. However, the system has one rather serious drawback. A high-amplitude, low-frequency variation in the output signal is produced, particularly when converting numbers in the middle range. This type of output signal is generally tolerable only in systems having a very long time constant.

The system which was finally developed retains the feature of allotting a discrete position for each pulse produced, yet does not develop the high-amplitude, low-frequency variation noted for the previous system. It approaches the ideal system mentioned earlier, in that the pulses produced for converting a given number are distributed relatively uniformly over the time period T. This system converts each individual binary digit into a proportional pulse rate.

The system is in several respects very similar to the analog weighting of digits conversion method discussed in the introduction, and represented schematically by Fig. 1. Referring to this figure again, if the voltage source is eliminated and individual current pulse sources substituted for the resistors, it becomes a simplified diagram of the new system. The current pulse source substituted for each resistor must produce a current which is equivalent to that produced by the resistor it was substituted for. Fig. 5 shows the pulse patterns obtained with this system for the conversion of some typical numbers. The pulse patterns generated for 1, 2, 4, or 8 have discrete locations relative to the period T. Furthermore, none of the pulses in the various trains coincide (timewise) with one another. This feature permits combining pulse trains to obtain all possible digital combinations. It will be noted that a 5 is produced by combining a pulse train for a 4 with the pulse train for a 1, and that a 13 is produced by combining the pulse trains for an 8, 4, and 1.

The method of generating pulse rates which bear a binary relationship to one another, yet have no pulses coincident with one another, is shown in the logic and timing diagram of Fig. 6. A simple, free-running binary counter is used, excited by some frequency source f (which will generally be the computer clock or some submultiple thereof). The outputs obtained from the flip-flops are assumed to be the differentiated positive going signals. The output from the first flip-flop (P_2) will be a pulse rate at a frequency $f/2$. Since the next flip-flop is being excited by a frequency $f/2$ which is out of phase with the output P_2, the output P_1 will be a pulse rate one-half that of P_2 and out of phase with P_2. The same reasoning applies to the next stage which will produce a pulse rate equal to one-half the preceding stage and out of phase with it, and applies all the way down the line for as many stages as one cares to consider. The number of such stages needed equals the number of digits (word length) of the binary number being converted. These binary related pulse rates are applied to gates which are enabled by the appropriate digit of the number being converted. The outputs of all the digit gates are combined on a common output line. At this point we have an average pulse rate proportional to the input binary number.

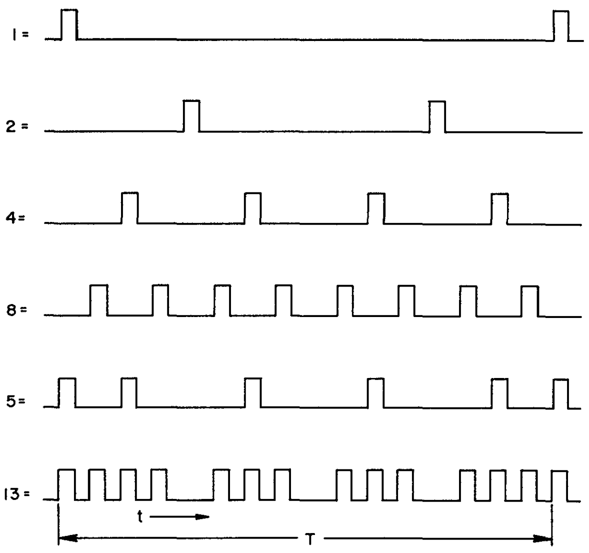


Fig. 5—"Binary weighted" pulse patterns.

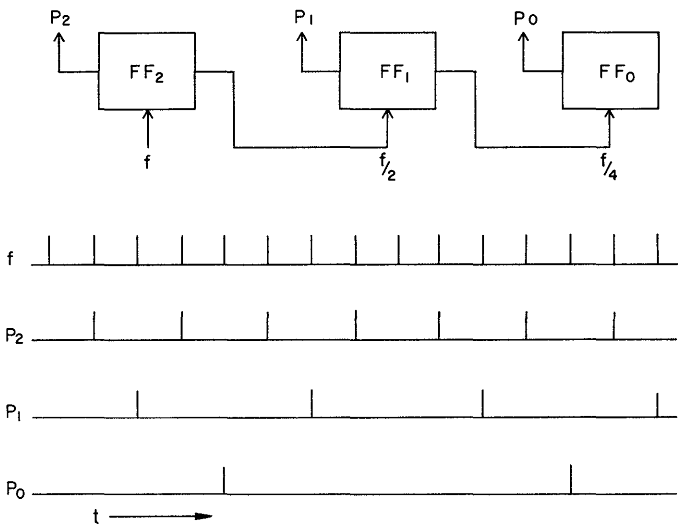


Fig. 6—Logic and timing diagram. Binary related pulse generation.

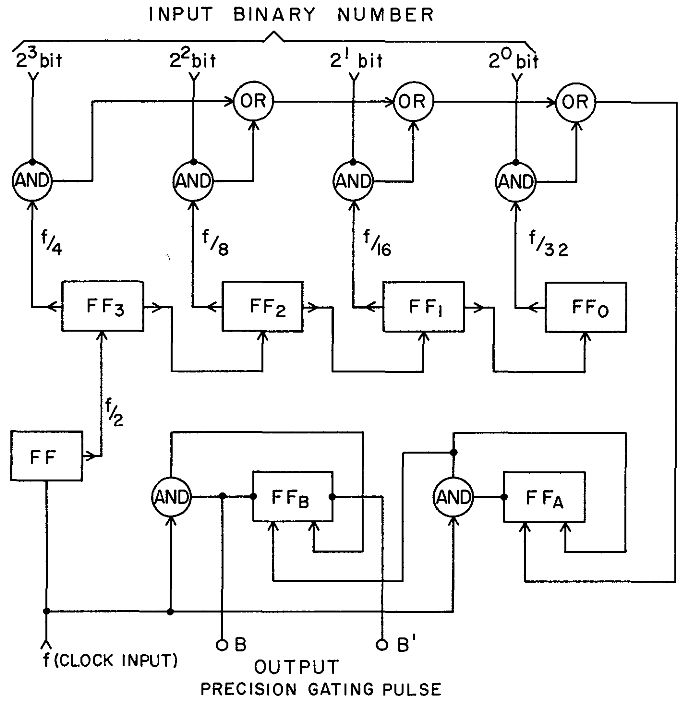


Fig. 7—Simplified 4-bit converter.

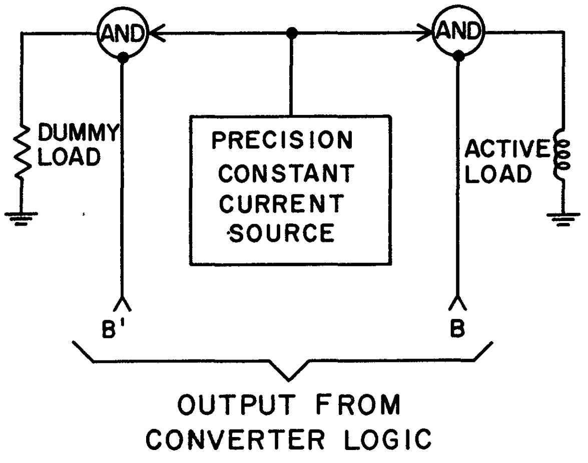

AND
DUMMY LOAD
PRECISION CONSTANT CURRENT SOURCE
ACTIVE LOAD
B'
B
OUTPUT FROM CONVERTER LOGIC

Fig. 8—Constant-current pulse generator.

Controlling the Pulse Width and Amplitude

The preceding section has dealt with the problems associated with generating a number of pulses in period T equal to the digital number to be converted. The pulse rate thus produced can be used to initiate a circuit which produces precisely gated pulses synchronized with the computer clock frequency. The turn-on and turn-off times of these gated pulses are controlled by computer clock pulses applied to a flip-flop. This description is clarified by referring to the simplified logic of the system, Fig. 7. A control pulse initiates the generation of a precision gated pulse by setting FF_A. This enables the associated AND gate and causes the next clock pulse to turn on FF_B, and reset FF_A. The next clock pulse turns off FF_B.

Thus, FF_B has been turned on for precisely the interval between two clock pulses. For the present discussion we will consider that the clock source is a stable constant frequency. Hence, the pulses produced will be of constant width.

This precision gating pulse is used to produce current pulses of a constant amplitude. This is accomplished by switching a constant current into the active load when the precision gating pulse occurs, or into a dummy load of approximately the same impedance when the gating pulse is not present. The circuit is shown in Fig. 8. BB' is the output of the converter logic (Fig. 7). Switching between the dummy load and the active load causes the loading on the constant current source to remain approximately constant, which greatly improves the stability of the current source.

Integrating and Smoothing the Pulses

One of the simplest methods of utilizing the precision pulses to produce an analog quantity proportional to the input binary number is to apply the pulses to a D'Arsonval meter winding. The time constant of the meter can be long relative to the applied pulse rate and it will effectively average the applied current. The analog output will be a meter reading (shaft position) proportional to the input binary number. In using this system for digital HIG gyro torquing, the same situation exists wherein the time constant of the device is long relative to the applied pulse rate. In this case, the output is an angular rate rather than a shaft position.

A more general method of averaging the pulses is to use an RC circuit. Common circuits employed for current smoothing and averaging—often loosely described as integrating circuits—consist of a capacitance shunted with a resistance. If the time constant of this circuit is large compared with the pulse train cycle time, the voltage across the capacitor will rise until the average flow of current into the capacitor is opposed by the average discharging current through the resistor. The mean value of the capacitor voltage is a measure of the mean rate of the precision pulses. The extent to which this voltage fluctuates (ripple content) will be governed by the choice of the RC time constant. As this time constant is increased, the ripple becomes smaller in amplitude. More sophisticated filters may also be employed to reduce the ripple content. As desired, the analog output may be taken as either a voltage or a current.

A method for obtaining a voltage or shaft position proportional to the input binary number using a RC integrating circuit is shown in Fig. 9. A voltage is produced at point A which is proportional to an input binary number. Conversion to a shaft position is obtained by mechanically driving a potentiometer to obtain an equal voltage. In the illustrated conversion to a shaft position, the stability requirements on the voltage source are greatly reduced because it is used as a standard by both the precision pulse generator and the potentiometer. A change in this supply voltage is reflected as a proportional change in the output of both the potentiometer and digital to voltage converter circuits. For precise digit-to-voltage conversion the voltage source should, of course, be more precise than the precision desired in the output.

Accuracy Considerations

The accuracy of this conversion system is determined primarily by three factors. First, the precision to which the ratio of on time to off time in the period T can be made equal (or proportional to) the number to be converted. Second, the precision to which the current can be switched completely on and completely off when desired. Third, the accuracy of the constant current source.

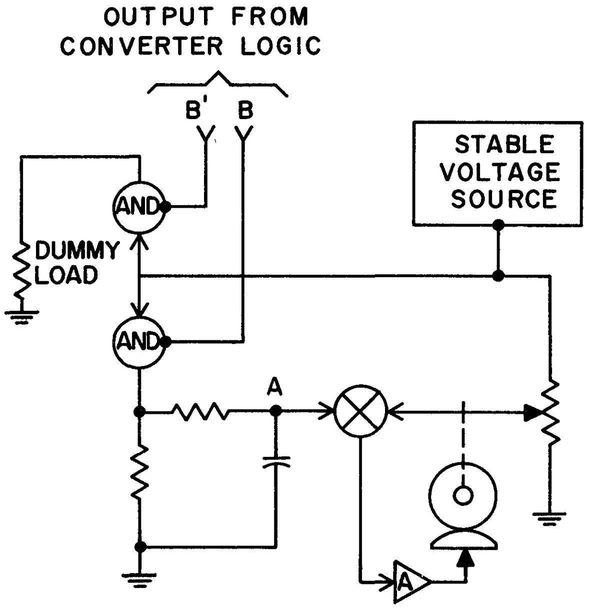


Fig. 9—Digit-to-voltage or shaft-position converter.

In considering the first requirement, it is interesting to note that the actual width of the standard pulse generated is not what must be controlled, but its relationship to the period T. This is because it is the ratio of the on time to off time of the pulse (and the number of pulses) that determines the analog output produced. By turning the pulse on and off with clock pulses, the ratio will be constant regardless of the clock frequency. The uncertainty as to when the pulse turns on and when it turns off, with respect to the initiating pulses is then the primary concern in determining the width precision of the pulses. With transistorized flip-flops, which trigger in approximately 0.1 microsecond, the uncertainty was determined to be less than 0.01 microsecond. The accuracy of the width of such pulses is then a function of the pulse width. With a one millisecond pulse width, the pulse width accuracy is one part in 100,000.

The details of the current gating circuits are beyond the scope of the present paper. However, it should be mentioned that it requires more than a simple gate to switch currents fully on and fully off. Circuits tested have shown on/off ratios of 1,000,000 and higher.

The primary factor limiting the accuracy of this system is the constant current source. The system was tested using a current source which maintained the current constant to one part in 20,000. Since the loading remains constant, this accuracy is not unduly difficult to obtain.

Conclusion

The conversion technique described is believed to be new. The utilization of this technique has resulted in a converter that has greater accuracy with fewer critical components than converters built using other known conversion techniques.

It is basically a parallel, straight binary-to-analog converter. The system is quite adaptable to time sharing. The frequency source and individual pulse-train generators associated with one conversion may be common to as many conversions as may be required. The additional circuitry for additional conversions is the gates associated with each digit of the binary number to be converted, and the circuits needed to standardize the pulses. Time-sharing techniques can be used in applying several different binary inputs to their associated gate enable lines. Capacitor "memories" on these lines eliminates the need for individual registers to hold each binary input.

Acknowledgment

The conversion system described was developed on a company-sponsored airborne digital computer research project. The author also wishes to acknowledge the suggestions and encouragement given by T. Lode which led to the development of this technique.

Discussion

W. A. Erickson: What was used as a constant current source?

Mr. Glick: The constant current source used in this system was transistorized current source built for an analog computing system.

W. Hochwald (Autonetics): It is not clear how the accuracy is independent of clock frequency.

Mr. Glick: The reason the accuracy is not affected by clock frequency is because of the fact that we are extracting a period of time T in which we wish to produce a certain number of pulses. Now, the output we get will, of course, be a function of the number of pulses that we put into this time period T and the ratio of the time the current is on to the time the current is off. Since we gear this time period to our basic computer clock which says that the time period T can vary, but the number will remain the same and therefore, the average current produced will remain the same regardless of variation. If there is a discontinuity in frequency, there will be an error. However, low velocity changes in frequency will not affect the accuracy of the system.

A Reliable Method of Drift Stabilization and Error Detection in Large-Scale Analog Computers

EVERETT E. EDDEY[†]

Introduction

THE NEED FOR continuous, automatic stabilization of the zero balance of the dc amplifiers in modern electronic analog computers is well established. Goldberg,[1] in 1950, described the first or one of the first successful circuits for this purpose. In 1951, Ingerson[2] described a drift-stabilization system using a single stabilizer amplifier in conjunction with a multichannel mechanical commutator to stabilize several dc amplifiers. This type of system has been used in the GEDA[3] L3 and N3 and other analog computers.[4]

Although the stabilization system used in the L3 and N3 computers is successful in its primary purpose of reducing drift, it was not designed to take full advantage of some of the inherent properties of commutator stabilization. These properties make possible highly reliable and dependable operation and provide accurate and immediate detection of errors.

The stabilization system used in the new GEDA A14 general-purpose electronic differential analyzer makes full use of these properties. GEDA A14 installations may contain several hundred dc amplifiers. A typical small-scale installation is shown in Fig. 1.

This paper describes the operation of the A14 stabilization system and error-detection circuits, and the methods chosen to improve reliability. Finally, the reliability and dependability to be expected from the A14 are discussed.

Drift-Stabilization System

The over-all operation of the A14 drift-stabilization system may be understood with the aid of Fig. 2. For simplicity, only a few of the commutator contacts are shown. Because the circuitry associated with each dc amplifier is essentially the same, only one amplifier is shown being stabilized. The dc amplifiers may be either standard computing amplifiers, switched amplifiers (such as used in electronic multipliers), dc servo-amplifiers, or dc power-supply-regulating amplifiers.

† Goodyear Aircraft Corp., Akron, Ohio.

[1] E. A. Goldberg, "Stabilization of wide-band direct-current amplifiers for zero and gain," *RCA Rev.*, vol. 11, pp. 296–300; June, 1950.

[2] W. E. Ingerson, "Drift Compensation in D-C Amplifiers for Analog Computers," paper presented at IRE National Convention, New York, N. Y., 1951.

[3] Reg TM, Goodyear Aircraft Corporation, Akron 15, Ohio.

[4] D. W. Slaughter, "Time-shared amplifier stabilizes computers," *Electronics*, vol. 27, pp. 188–190; April, 1954.

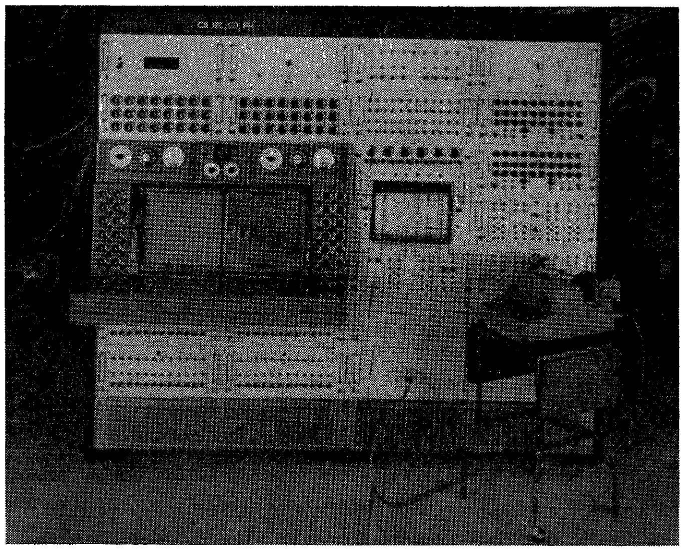


Fig. 1—Typical small-scale GEDA A14 installation.

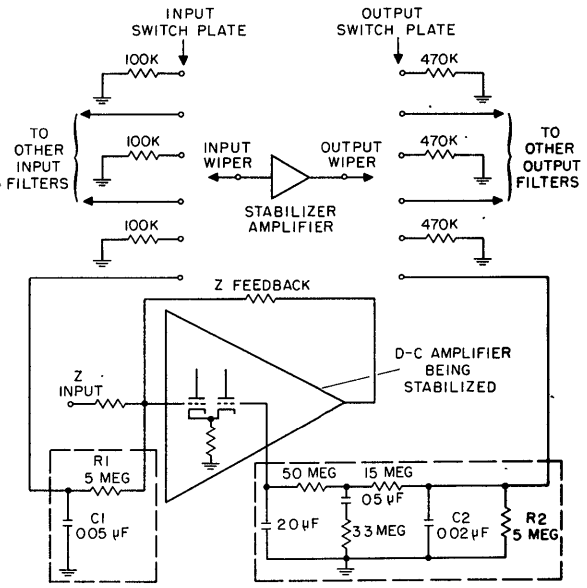


Fig. 2—Major parts of drift-stabilization system.

Hence the entire computer can be drift-stabilized by this system.

The signal grid voltage of the dc amplifier is fed to a commutator contact through the input filter. The input filter removes signals that are synchronous with the commutator sampling rate; otherwise, these voltages would be synchronously rectified by the commutator and appear as offset in the dc amplifier. The voltage applied to the commutator contacts is sampled by the commutator input wiper and amplified by the negative-gain stabilizer amplifier. This amplified voltage is then sent through the output wiper to the output filter of the same dc amplifier. The output filter reduces the ripple resulting from the pulse voltages received from the commutator. The filtered voltage is applied to the balancing grid of the dc amplifier, where it acts to restore the dc balance of the amplifier.

The commutator, the stabilizer amplifier, and the input filter capacitor, C1, are located in the same A14 modular plug-in unit (see Fig. 3). The other components of the input filter, along with the output filter components, are located with the associated dc amplifier. In Fig. 3, the commutator switch end-bells have been removed to show the ease of cleaning the contacts.

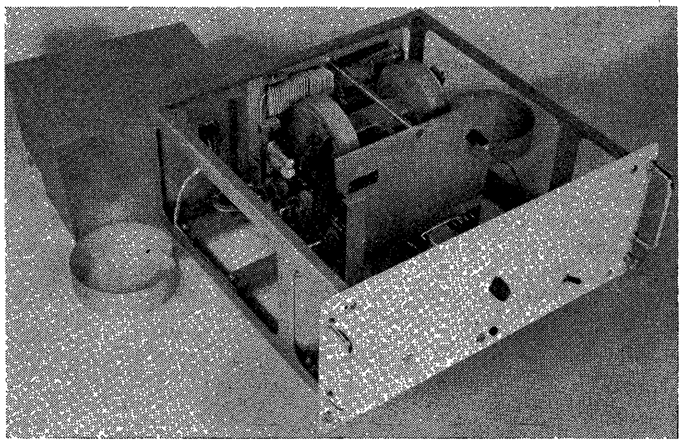


Fig. 3—GEDA A14 stabilizer modular unit.

Error-Detection Circuits

The commutator stabilizing system provides a convenient central point for originating signals indicating faulty operation anywhere in the computer. Almost all component failures affecting the operation cause abnormally large voltages at the signal grid of the associated dc amplifier unit. The resulting large pulse from the stabilizer amplifier triggers the warning indicators.

Three types of warning indicators are used in the A14 computer: 1) a master indicator, on the computer control panel, that shows faulty operation and which, if the operator desires, will stop the computer and hold all voltages; 2) an easily visible group indicator that signals the general location of the offending element; and 3) individual indicators that pinpoint the defective unit. For example, if a tube in one of the switched dc amplifiers in an electronic multiplier becomes defective and impairs the accuracy of the solution, the master indicator on the computer control console will light, the group indicator on the particular electronic multiplier modular unit will light, and the individual light for the defective multiplier amplifier will light.

Individual Indicators

The operation of the individual indicators can be understood with the aid of Fig. 4. The NE-2 neon lamp is connected to the input of the output filter of the dc amplifier with which the indicator is associated. Resistor R2 and capacitor C2 are the same as in Fig. 2.

The NE-2 lamp sets the threshold value at which a pulse from the stabilizer will turn on the indicator. The NE-51, mounted on the front panel, provides the visual indication. The low-impedance positive bias source is set at a value between the firing and extinguishing potentials of the NE-51, which must be selected for these potentials. A built-in selfchecking circuit tests the NE-51 lamps.

A voltage pulse that fires the NE-2 will also raise the voltage across the NE-51 to the firing voltage. Once the NE-51 is ignited, current from the bias supply maintains the ionization. Capacitor C3 and crystal diode 1N34 form a diode clamping network permitting operation on both positive and negative pulses from the commutator. The NE-51 thus provides a memory-type indication that remains until turned off by the operator. Reset is accomplished by inserting high resistance in series with the bias supply so the current is reduced below the value necessary to maintain ionization. All NE-51 lamps are extinguished simultaneously. High resistance may be left permanently in series with the bias supply if a nonmemory flashing-type indication is desired.

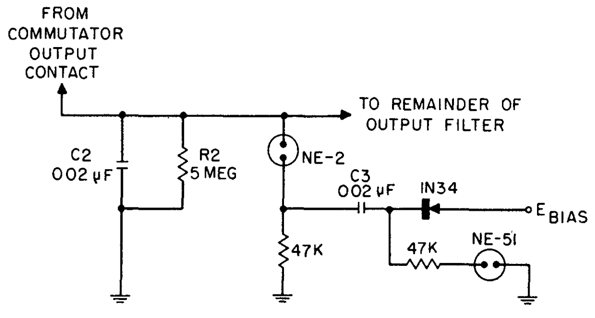


Fig. 4—Individual warning indicator.

Group Indicator

The group indicator is activated by the individual indicator circuits. All the NE-51's in a given modular unit, instead of being connected directly to ground as shown for simplicity in Fig. 4, are connected to a transistor flip-flop circuit. The input impedance of this circuit is on the order of a few hundred ohms; thus the operation of the individual indicator circuits is not affected. The flip-flop circuit operates a relay controlling an incandescent light on the front panel. This light is easily visible from a distance. Fig. 5 shows the front panel of an electronic multiplier modular unit with individual and group indicators.

Master Indicator

The signal for the master indicator is derived from a small resistor in series with the bias supply for the individual indicators. When any NE-51 fires and draws current from the bias supply, the voltage across the resistor changes. Amplified, this voltage change operates a relay that energizes circuits controlling the master

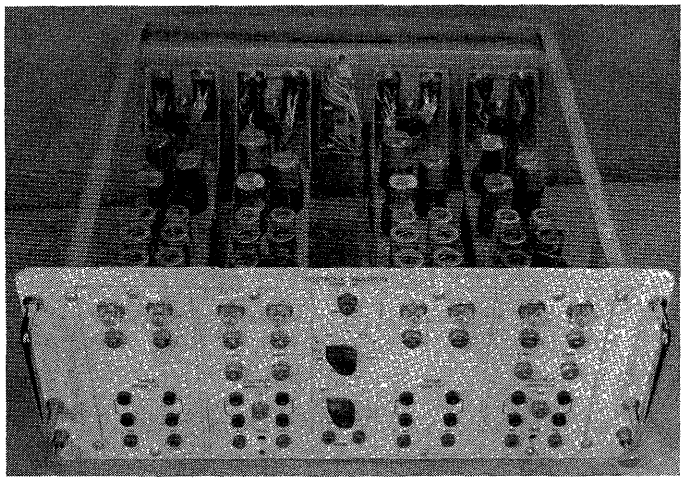


Fig. 5—Front panel of GEDA A14 electronic multiplier.

indicator lamp. At the operator's choice, the relay in addition will energize an audible alarm, and place the computer into "hold," stopping the problem solution.

Self-Checking Circuits

The stabilization system itself has two built-in self-checking circuits. The first provides a continuous check on the operation of the commutator and stabilizer amplifier. A small dc voltage is applied to a contact of the input pole of the commutator. The resulting amplified positive output pulse is applied through a filter to the grid of a tube having a large, fixed negative bias. This pulse is normally sufficient to keep the tube conducting strongly. If the pulse is absent or reduced in value (commutator stopped, stabilizer amplifier gain reduced, etc.), the tube will be cut off. The resulting high plate voltage flashes a neon lamp through a relaxation oscillator circuit. The neon lamp gives visual indication on the stabilizer modular unit; at the same time, the master indicator is activated. The second test circuit may be switched on occasionally to check commutator leakage. A dc voltage on the order of 10 v is substituted for the ground potential to which the 100-k and 470-k isolating resistors are normally returned. An oscilloscope is connected to each wiper arm in turn. If there is no leakage, the oscilloscope display will be a series of regular pulses. If there is leakage in any contact, it will be indicated by a change in pulse height. An oscilloscope pattern showing a contact with leakage is presented in Fig. 6.

Improvements in A14 Stabilizer Reliability

Major factors affecting the reliability and operation of the A14 stabilization system, in approximate order of decreasing importance based on experience with previous computers using commutator stabilization, include: leakage between adjacent contacts on the commutator, phasing of the commutator, mechanical failure of the commutator, hum pickup, and capacitive cou-

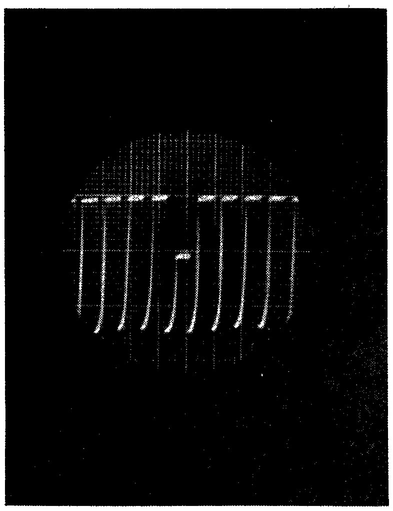


Fig. 6—Oscilloscope test waveform.

pling between input and output sections of the commutator. Methods used to improve these factors will now be described.

Many of the items affecting reliability in the past were associated with the commutator itself, probably because the commutators used in the early computers were originally intended for telemetering applications. These applications required sharp edges on the switching pattern, with relatively short life being acceptable. These conditions are not applicable in drift stabilization systems where sharp transition from switch "off" to switch "on" is unimportant. Consequently, investigation was made, in association with the commutator manufacturer, of the proper design of a commutator for exclusive use in drift stabilization.

Leakage between adjacent input contacts on the commutator causes both cross talk between channels and reduced gain, while leakage between adjacent output contacts causes only reduced gain. The input-contact cross-talk effects are greatly reduced by the 100-k isolating resistors and are not a major problem. However, leakage can reduce the stabilization gain almost to zero, and this effect has been the principal source of stabilization system failure in the past.

Leakage between contacts is caused by the accumulation of carbon particles from the silver-graphite brushes of the wiper arms. Experimentation has shown that life can be improved by reducing the brush pressure against the contacts. This pressure, in the case of the commutators intended for telemetering use, was more than 15 ounces. Satisfactory switch action for use in the stabilization system can be obtained with brush pressures of just a few ounces.

Optimum service-free brush life requires a fairly careful control of brush pressure and composition for controlled release of carbon-wear particles. Release of some particles is essential for proper lubrication. However, the particles must be small and released slowly to delay the build-up between adjacent contacts. The actual wear on the brush is quite small, with the ultimate brush life, based on the wearing away of material only, probably being about 50,000 hours. The service-free life of the brush and commutator is, of course, much shorter because of the development of leakage. Tests indicate an average service-free life of 5000 hours for the present commutator. Investigations are being made of methods of continually removing the wear particles from the commutator (*e.g.*, by use of an air scoop). If these methods are successful, the life of the commutator may be limited only by the ultimate life of the brush.

Ease in cleaning the switch contacts has been provided by designing the commutator to make the contacts accessible by merely removing a dust cover, as shown in Fig. 3. Cleaning can be completed in a few minutes without disturbing any of the commutator adjustments. The construction also facilitates the measurement, and hence the control, of brush pressure.

Both the phasing and leakage requirements on the commutator are reduced by the type of output filter used (Fig. 2). The output wiper is phased to always lead the input wiper. A signal is sent from the signal grid to the output filter contacts only when both input and output wipers are simultaneously contacting signal contacts (including the shorting time to the adjacent contact). This interval is the channel "on" time. The charging-time constant of C2 with the output impedance of the stabilizer amplifier is considerably less than the average channel "on" time. Thus variations in channel "on" time produced by phasing differences result in small changes in stabilizer gain. A change of 25 per cent in the "on" time results in a gain reduction of less than 4 per cent.

The rectification efficiency of the output circuit (the ratio of the dc voltage developed across C2 to the output pulse amplitude) would be nearly 100 per cent if it were not for R2. R2 lowers the rectification efficiency to about 20 per cent by tending to discharge C2. The gain of the stabilizer amplifier is increased by a factor of 5 to compensate for this lower efficiency. Also, the effects of commutator leakage resistance, which is essentially in parallel with R2, is somewhat reduced, and the higher amplifier gain is useful in the error-detection circuits. The stabilizer gain is still an order of magnitude less than that used in the GEDA L3 and N3 stabilizer amplifier in which the holding capacitor C2 is not used.

Mechanical failures of the motor and bearings used in the earlier commutator can also be largely ascribed to the use of components not necessarily intended for long life. With a commutator specifically designed for long life, these failures should be almost eliminated. Capacitive coupling problems are largely eliminated because the stabilizer amplifier gain has been reduced.

Hum pickup in the input circuitry results in low-frequency variations, on the order of magnitude of the hum, in the output of the dc amplifier being stabilized. One of the principal hum components can be largely filtered out by placing C1 in the commutator instead of in the dc amplifier. This procedure reduces the effect of hum voltage induced in the loop from the dc amplifier signal grid to the commutator and back through the ground. Much more freedom in cable layout is thus afforded.

System Reliability and Dependability

The reliability and dependability of the A14 drift-stabilization and error-detection system can be discussed under three aspects: mean time to failure, per cent service, and dependability itself.

Mean Time to Failure

The reliability of any system is indicated by the length of time it will operate properly without a breakdown. Computation of the mean time to failure of the system, a commonly used index of the capability of equipment to resist failure, permits quantitative description. The mean time to failure, of course, yields no particular information regarding the time between any two successive failures. The latter time is a function of the time distribution of failures of each component.

In any discussion of drift-stabilizing systems, the question arises of the relative reliability of a commutator system vs a system using chopper-stabilized amplifiers. Table I lists major items required in the A14 commutator system, with mean time to failure of each.

TABLE I

Components of Commutator Stabilizing System

Component	Quantity	Mean Time to Failure—(hour)
Commutators	4	5,000
Tubes	12	15,000
Resistors (composition)		
Under load	92	10^6
No load	2,670	2×10^6
Capacitors	1,326	10^6
Transformers	4	10^6
Selenium rectifiers	4	25,000
Transistors	8	25,000
Silicon diodes	8	25,000

System Mean Time to Failure = 194 hours

Table II gives similar data for one of the more reliable chopper-stabilized systems currently available. The quantities are those required in a fairly large-scale computer currently being fabricated by Goodyear Aircraft Corporation. This computer uses 329 dc amplifier channels. The transformers, selenium rectifiers, and transistors listed in Table I are required in the regulated dc filament supply for the first two tubes of the stabilizer amplifier. Values for tube life given in Table II are greater than in Table I since an ac amplifier is used in the chopper system. The mean-lives of all components except the commutator are based on data taken over a four-year period in Goodyear Aircraft's Dynamic Systems and Computation Laboratory, and on published data.[5,6] Information on commutator life is given elsewhere in this paper.

TABLE II

Components of Chopper-Stabilizing System

Component	Quantity	Mean Time to Failure—(hour)
Choppers	165	10,000
Tubes	329	20,000
Resistors (composition)		
Under load	987	10^6
No load	3,290	2×10^6
Capacitors	2,303	10^6

System Mean Time to Failure = 26.4 hours

Per Cent Service

By per cent service is meant the ratio of actual operating time to operating plus trouble-shooting and maintenance time. High per cent service is just as important as long failure-free life in computer applications, where equipment breakdowns may be annoying but do not affect safety or completion of a mission.

The A14 system is unusually well-suited for high per cent service because the operation of 83 stabilization channels can be checked simultaneously from one location. There is no need for individual checking of a number of chopper-stabilized amplifiers to determine the defective units. The shortest-lived component in the A14 system as shown in Table I is the commutator. However, the life figure given is that for failure due to switch leakage; the ultimate life is much longer. Cleaning of the contact plate requires just a few minutes. The built-in checking circuits show when cleaning is needed.

Dependability

The dependability of a system represents the confidence the operator can have that the system is operating properly at any particular time. The error-detecting circuits previously described give the operator assurance that the computing equipment is functioning at all times. Another paper[7] describes the A14 problem analyzer system. This system gives the operator a check on the accuracy of his solution. Together, these systems should give the operator the highest confidence in the proper and accurate operation of the A14 computer.

[5] R. L. Wendt and M. H. Smith, "A bombing system reliability program," 1956 IRE Convention Record, part 6, pp. 68–74.

[6] V. Harris and M. M. Tall, "A Progress Report on Reliability Measurement and Prediction," paper presented at Second National Symposium on Quality Control and Reliability in Electronics, Washington, D. C.; January, 1956.

[7] W. C. Meilander, "A new method of verifying analog computer problems and performance," this issue, p. 138.

Conclusion

Improvements in the A14 commutator system have produced a reliable method of drift stabilization that gives convenient indications of malfunction anywhere in the computer. The system offers high per cent service and high dependability. The writer would like to give acknowledgment to his following associates at Goodyear Aircraft Corporation: P. J. Hermann, R. Armstrong, and W. H. Byers for data used in calculating mean time to failure, and to C. D. Morrill and S. B. Yochelson for their fine cooperation.

Discussion

S. Rogers (Convair): Is it true that the A14 commutator system improvements are in mechanical features and accessory circuits rather than in the basic circuits, or is there new art here, too? Do you have available reports on your chopper, commutator, and tube failures and on your failure-record system?

Mr. Eddey: Partly yes and partly no. The commutator permits easier cleaning of the unit and also better control of the pres sure. We have used improved bearings and better motors, necessary to improve the mechanical operations of the unit. The circuitry we have improved by decreasing the requirements on the commutator itself.

A New Method of Verifying Analog Computer Problems and Performances

WILLARD C. MEILANDER[†]

Introduction

DURING the past decade the electronic differential analyzer has become an effective research and development tool. Present techniques allow the computer user to translate nearly every conceivable physical[1] system into computer wiring diagrams.

The analog computer ideally is a super slide rule, for while it is easily capable of arithmetic operations, its outstanding ability to handle integrodifferential operations is its most salient feature. The conventional slide rule has been accepted as a reliable device, but the differential analyzer has not been so universally received. In fact, one of the more difficult problems associated with differential analyzers is determining whether they are solving the desired mathematical relations. The obvious technique of point-to-point verification of the wiring diagram is far too time consuming to be practical for most problems; therefore, several necessarily cumbersome methods have been developed to ascertain when the computer is correctly wired for a given problem. Some of these methods, such as digital or analytic solution of a test case of the general problem, are very laborious and time consuming, and, if the analog solution of the test case does not agree with the analytic or digital solution, the location of the error or errors still is not known.

The problem then is: How can the reliability of the electronic differential analyzer, a super slide rule, be made more comparable to that of its simpler predecessor?

In the early days of analog computer use, the problems were comparatively simple. The small number of amplifiers and computing elements employed for a given problem did not require an elaborate method for determining whether the correct hookup was made on the problem board or whether the elements of the computer were operating correctly. Today, analog computers are becoming larger and more complex. The number of elements used for a given problem may exceed several hundred. Checking the computer to determine that the components are operating satisfactorily and, more important, that the problem is correctly wired has become a bothersome task. Present techniques require a point-by-point check of all elements of the system with an ohmmeter or by visual observation, or both. Generally, after the problem board is wired by one operator, a second operator checks it visually and compares it with a block diagram of the desired wiring.

The individual scales of a slide rule are checked by matching the indexes correctly for a particular problem. The result is accepted without question. The differential analyzer can be checked in a similar manner by verifying each computing operation, including scale, and comparing the hookup with the desired wiring diagram or directly with the desired mathematical relations. Once it has been established that the requested computations

[†] Goodyear Aircraft Corp., Akron 15, Ohio.

[1] The scope of problems studied is not, of course, limited to physical problems. Much important work in economic and other non-physical fields has been carried out with the differential analyzer.

are not beyond the capabilities of the computer, it can be more effectively used as a research and development tool than its simpler relative, the slide rule, because of its high accuracy, speed, scope of operations, and flexibility.

The GEDA[2] A14 computer readily copes with these difficulties of problem analysis by using a system that readily provides a complete printed record of the wiring system and its individual components. The problem-board hookup can be verified exactly; that is, each amplifier and each computing element can be checked to determine that it is correctly connected to the next computing element in the wiring. The A14 problem-analyzer system also indicates whether the element under test is hooked up as a summer or as an integrator and determines the over-all gain setting of the element in question.

As is shown in Fig. 1, the over-all gain, K, involves all the computing circuits associated with the amplifier for a given problem. The computer network, as it is hooked up with its coefficient potentiometer, its input resistor, and its feedback circuits, must provide a gain setting to fit the requirements of the problem.

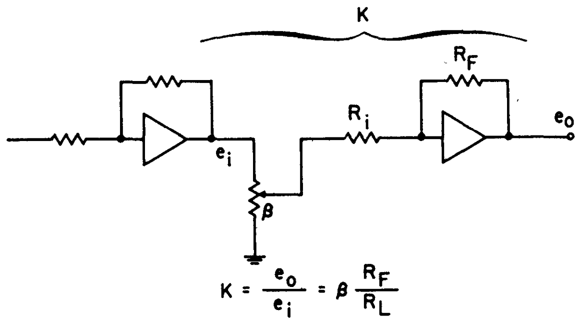


Fig. 1—Elements involved in determining the over-all gain of a computer amplifier.

The A14 problem-analyzer system easily determines the correct gain setting by the method shown in Fig. 2. A single-pole double-throw relay is located at the output of each amplifier and at each of the other computing elements. This relay will switch the output of the amplifier to an equivalent test-output voltage which, in turn, is applied as a substitute for the output of amplifier $\underline{1}$.[3] When the output-monitor switch is stepped to position $\underline{2}$, the output depends on the feedback and the input circuits of amplifier $\underline{2}$, thus providing a complete check on the over-all gain of amplifier $\underline{2}$. Similarly, when the monitor switch is stepped to position $\underline{6}$, the output of amplifier $\underline{6}$ represents the over-all gain produced by that part of the system from the equivalent output of amplifier $\underline{1}$ to the output of amplifier $\underline{6}$, including the effects of the feedback resistor, the input resistor, and the coefficient potentiometer. An integrator or any of the nonlinear elements of the A14 computer may be checked by using the A14 problem-analyzer system.

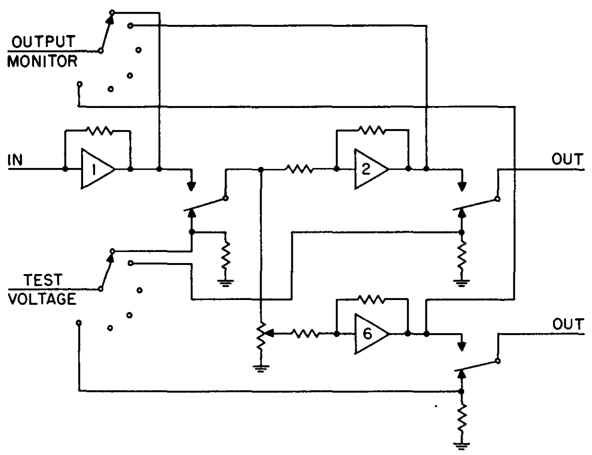


Fig. 2—Simplified A14 problem setup.

By checking the network feedback, input resistors, and coefficient potentiometers in this manner, the amplifier performance also is verified. A comparison of results obtained by the problem analyzer with a block diagram of the desired wiring quickly reveals whether the problem is correctly wired and pinpoints the location of any errors.

In actual operation the checkout procedure is somewhat different because the output monitor switch automatically scans all outputs and records the output of each element on an automatic printer, providing a record in digital form for comparison with the wiring diagram. For a typical system involving 100 amplifiers and 20 nonlinear components, it would be possible to check completely not only the wiring but also the machine components and the gain setting of each of the computing elements in about 40 minutes of computer time. When compared to the time it takes an operator to perform the measurements required by any other method of problem analysis, it can be seen that the A14 problem analyzer offers an important service to the computing field. Moreover, the analyzing operation is not dependent on arithmetic. The operator does nothing but examine the printed results. All gain settings are recorded in their exact values by the printer, except those of the integrator circuits. For these, the printed gain settings must be multiplied by 10.

Another desirable feature of the GEDA A14 problem analyzer is its ability to set accurately the potentiometer values within the networks in which they are to be used. All GEDA A14 potentiometers can be set by using a voltage from the potentiometer itself or by using a voltage from the amplifier to which that coefficient potentiometer is connected. This permits a check on the over-all gain, which is the important factor in setting

[2] Reg. TM, Goodyear Aircraft Corp., Akron 15, Ohio.
[3] All figures or words capitalized and underscored in the text indicate designations on A14 equipment.

up any given problem. To obtain the correct potentiometer setting for the conditions presented in Fig. 2, the operator would turn the computer program control to the CALIBRATE position and introduce an equivalent signal for amplifier 1. He would then select output position 6, by dialing a two-digit code, and adjust the potentiometer until the output of amplifier 6 reaches the desired value when the precisely known test voltage of the input is considered. Nearly all system errors have been eliminated. Errors in the test-input voltage and the output-monitor voltmeter can be adjusted to less than 0.01 per cent; therefore, the over-all gain setting can be adjusted to within 0.01 per cent of the desired value.

Another significant feature of the A14 analyzer is that while all voltage measurements are made at points of low impedance, all high-impedance elements, such as input resistors and potentiometers, are measured very accurately. Measuring at high-impedance points often introduces hum, cross talk, and signal noise that obscure desired results.

The automatic checking system operates as follows. The problem board to be checked is inserted into the A14 computer. The PROGRAM control is set to CALIBRATE, and all coefficient potentiometers are calibrated; then, the system is set to automatically scan all outputs, and a sequence of operations is performed to check the connections from the output of each amplifier. In the automatic problem-analyzer mode, all relays at outputs of computing elements are turned to the ANALYZE position and an equivalent input is introduced for amplifier 1. The output-monitor switch then automatically scans all computing element outputs; when it finds an element with a voltage greater than 100 mv, it stops, and the voltage is read on a digital voltmeter. After the digital voltmeter has stabilized, the printer records the following information: 1) the test-voltage origin, 2) the magnitude of the test voltage, 3) the amplifier at which the output is read, and 4) the scale factor, or gain, of the amplifier. The printer then releases the output-monitor switch to search for the next computing element with an output exceeding 100 mv. The operation may be repeated until all outputs have been checked.

For the conditions presented in Fig. 2, for example, the printer would designate amplifier 1 as the test-voltage origin, designate amplifier 2 as the point at which the output voltage was read, record the value of the test-input voltage to amplifier 2, and record the gain of amplifier 2. Then the output-monitor switch would move automatically to position 6, print the amplifier number and the gain to this point, and as soon as the printed cycle was completed for each output, the scanner would move to the next position or to the end of its scan; on reaching the end of scan it would cause the input test voltage, as shown in Fig. 2, to move to amplifier 2 thus providing an equivalent output for amplifier 2. The automatic-scan cycle would then be repeated. In this way all amplifier outputs are checked, and an accurate digital record of the gain of each amplifier with its relationship to other elements in the system is provided. The reference voltage is removed from the board during most of the problem analysis, but it is applied for one test-input position. A scan of the outputs also shows where connections are made to the reference voltages.

When the scan cycle has been completed, the values of all initial conditions have been checked. Since the printed record is available to the operator in about 40 minutes for checking, the computer is not tied up for long periods for a single problem analysis. The problem board is removed from the analyzer so that the operator may conveniently check it with the printed record and make any corrections indicated. If it is desirable to store a problem board which has been analyzed in the A14, the checking operation can be repeated. The second printed record then can be compared with the earlier analysis to ensure that wires have not been changed and that potentiometer settings are correct.

Problem-board errors, as pinpointed by the A14 problem analyzer, can be altered in a short time. For example, if the second analysis of a problem board indicates no output for amplifier 2 (Fig. 2) when an equivalent output exists at amplifier 1, the operator knows immediately that either the lead between the output of amplifier 1 and the input of amplifier 2 is broken or that amplifier 2 is not functioning properly. In earlier systems the operator could only check the second-run results of the stored problem against the earlier results and, if these did not agree, a complete check of the system would be required to find the error.

The A14 problem analysis also can be performed manually; that is, a test voltage can be substituted for the output of amplifier 1 by dialing input 1; the output monitor can be switched to amplifier 2 by dialing output 2 and the printer can be operated manually by pushing the MANUAL-PRINT button. The dial setting of the potentiometer at the input of amplifier 6 can be typed in manually, using the electrically controlled typewriter, if it is desired to provide a reference to a particular position on the printed record so that the previous potentiometer setting can be readily discerned. It must be remembered, however, that the potentiometer dial is accurate only to three positions and that maximum accuracy can be obtained by setting the potentiometer to the approximate position, then adjusting the setting to provide the desired output. Where accuracy of 0.01 per cent is not required, the setting can be made manually and checked for gross errors.

After all the elements of a problem have been checked statically, and all input networks and coefficient potentiometers have been adjusted in the CALIBRATE position, a check is made of the dynamic performance of the integrator elements. The system is switched to the DYNAMIC TEST position, and a voltage is applied simultaneously to the input of all integrators. The outputs of all integrators rise at a controlled rate, then are shut off. The automatic scanner will detect the elements that are connected as integrators and can provide a printed record of these elements in one operation. The element numbers then can be checked against the elements shown as integrators in the wiring diagram.

The dynamic check completes the operation performed by the A14 problem analyzer. Thus it can be used to determine that all connections are as shown on the wiring diagram and that no extraneous connections exist. It checks the over-all gains of the system. It accurately checks potentiometer settings and each of the machine components of the system. It can be used to determine whether the proper initial conditions are applied.

An example of the typical information as printed by the A14 problem-analyzer equipment is as follows:

```
01       10
     02       −2.560
     12       −5.000
02       10
     03       −5.000
     07       −0.9999.
```

These data show that, reading from left to right, the equivalent signal for the output of amplifier 1 is 10 v, that a connection exists from the output of amplifier 1 to the inputs of amplifiers 2 and 12, that the gain of amplifier 2 is −2.560 and the gain of amplifier 12 is −5.000. The second grouping of data, as recorded by the A14 printer, tells the operator that the equivalent signal for the output of amplifier 2 is 10 v, that a connection exists from the output of amplifier 2 to the inputs of amplifiers 3 and 7, and that the gain of amplifier 3 is −5.000 and the gain of amplifier 7 is −0.9999.

The A14's automatic scanner also can be used in the program control positions STANDBY, I.C. (initial condition), and HOLD. In STANDBY the scanner will detect computing element failure by indicating the presence of an undesired output voltage. In I.C. the scanner will stop at the output of each amplifier where an initial condition exists. In HOLD the scanner will indicate all voltages at the outputs of every element in the machine unless these are negligible. The operator can obtain a digital record of his problem at any time because switching the system from OPERATE to HOLD causes all computing elements to retain the voltages that existed at the time of switching. A part of the record of a typical problem printed with the equipment switched to the HOLD position is as follows:

```
01        08.25
02        55.66
03        00.99
06        27.50
08        99.20
43        17.66.
```

This information shows the output voltage of each of the elements, listed in the column on the left, at the time the problem was switched. The problem is continued as if the interruption had not been made when the equipment again is switched from HOLD to OPERATE.

Among the several advantages of the A14 computer that assist the operator in setting up a given problem and assure him that the computer is functioning correctly are the following:

1) The A14 system provides for a change of time scale in a problem by a factor of 10, thus providing the operator with a check on the response characteristics of the problem as set up in the computer.
2) The system provides for the reversal of initial conditions in the problem without wiring changes and thus provides the operator with a check on the linearity of the computing elements for both polarities of output. This is useful in checking special circuits containing diodes.
3) The inherently low differential-operate time (less than 100 μsec) of all HOLD relays assures the operator that errors will not accumulate rapidly if the machine is switched from OPERATE to HOLD repeatedly during the running of a problem.
4) The A14 computer high-speed UTILITY RELAYS can be operated from the control position, or can be operated directly from the output of an operational amplifier.

Fig. 3 shows the internal construction and wiring of the side control panel. The major portion of the problem analyzer, the PROGRAM switch, and other control functions are contained in this control panel. Fig. 4 shows the A14 problem board and control panels from the operator's position. The computer operator who uses the A14 problem-analyzer system can be certain the problem is properly wired, that the computer components required for the problem are functioning properly, and that the proper scale factors have been applied. He can obtain this and other useful and accurate

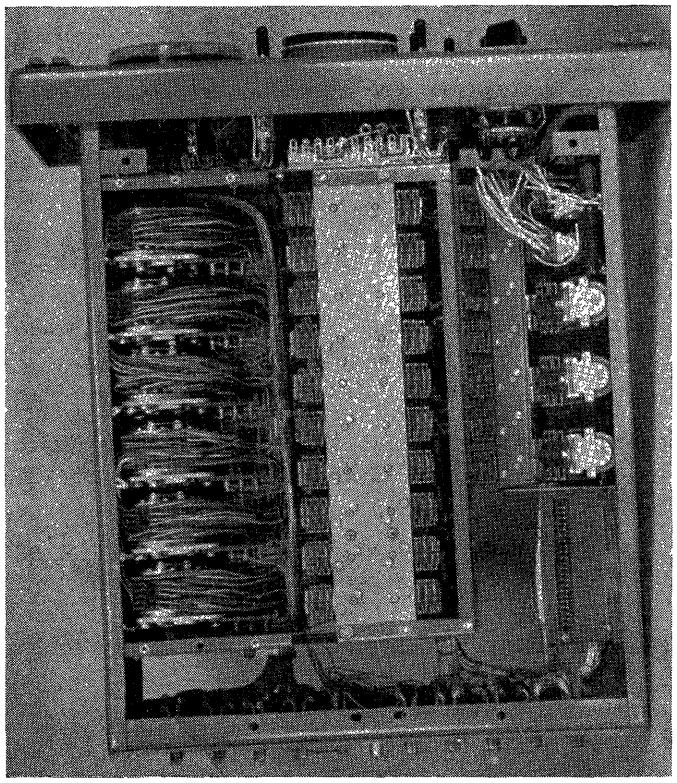


Fig. 3—Internal view of the A14 side control panel.

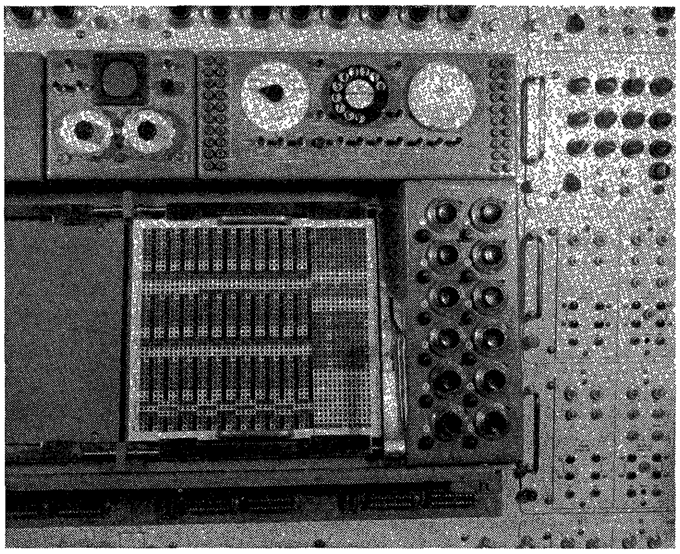


Fig. 4—Problem board and control panels of the GEDA A14 problem analyzer.

data for his problem analysis in digital form directly from the equipment in a relatively simple operation requiring only about 40 min. on a typical computer. The A14 problem analyzer operator, knowing his equipment is reliable and his problem properly set up, can be more confident of his results. He need not rely excessively on his own judgment of the reasonableness of a problem solution. He is relieved of laborious point-to-point checking and of the time-consuming transference of computer data to usable form. Thus the A14 marks a decided advancement toward greater computer reliability, a factor which has limited the extensive use of electronic differential analyzers since their inception.

Discussion

E. W. Purcell (Douglas Aircraft Corp.): How are the test voltages used in your problem analyzer mode determined, and how are they set?

Mr. Meilander: The test voltages are the output of regular operational amplifiers, specifically designated for this purpose. They are determined from our reference sources and driven from the reference sources, so that they will also have the same characteristics as reference the voltage chains by a very small amount over a long period of time. The setting of the amplifier voltages is accomplished by precision resistors which can be adjusted to the correct value.

The Lincoln TX-2 Computer Development*

WESLEY A. CLARK†

Introduction

THE TX-2 is the newest member of a growing family of experimental computers designed and constructed at the Lincoln Laboratory of M.I.T. as part of the Lincoln program for the study and development of large-scale, digital computer systems suitable for control in real time. Although, in general characteristics and design philosophy, it owes a great deal to its predecessors, Whirlwind I and the Memory Test Computer, the Lincoln TX-2 incorporates several new developments in components and circuits, memories, and logical organization. It is the purpose of this paper to summarize these new features and to give some idea of the historical development and general design objectives of the TX-2 program. Fig. 1 shows TX-2 in its present development stage.

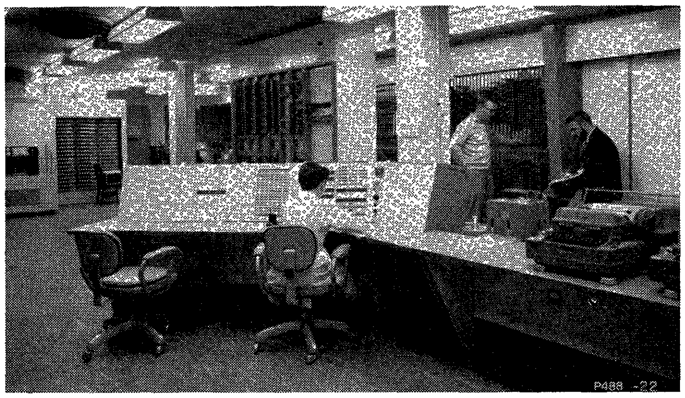


Fig. 1—The Lincoln TX-0 and TX-2 computers. Foreground: TX-0 console; middle center: TX-0 central computer frame; right rear: partially completed TX-2 frame showing plug-in unit construction; left rear: the 256×256 memory.

History

With the development by Lincoln and IBM engineers of the SAGE computer for air defense, real-time control computer systems had reached an impressive level of size, sophistication, and complexity. The highly successful 64×64 coincident-current, magnetic-core, memory array was in operation in the Memory Test Computer which had given up its earlier 32×32 array to Whirlwind. Vacuum tubes abounded in all directions. It was apparent that the further advances in system design which could be made by increasing memory size, eliminating vacuum tubes wherever possible, and organizing input-output buffering, control, and communications into more efficient forms, would be well worthwhile.

The development of a 256×256, switch-driven, magnetic-core memory array was begun and the Philco surface-barrier transistor made its appearance. After some very promising bench experiments with flip-flops and logic circuits, it became apparent that this transistor was potentially well-suited to use in large-scale systems and warranted further study. Accordingly, plans were laid for a succession of experimental digital systems of increasing size and complexity which would make possible the development and evaluation of circuits using the surface-barrier transistors, and which would lead to a computer of advanced design that would be capable of making efficient use of the 256×256 memory.

A double-rank shift register of eight stages and containing about 100 transistors was constructed and put on life-test in April, 1955. It has since been circulating a fixed pattern almost continuously with no known errors and no natural transistor failures.

As the next step, it was decided to build a small, high-speed, error-detecting multiplier and incorporate marginal checking and other system features. The value of a multiplier as a preliminary model had been well demonstrated by the 5-digit system built during Whirlwind's early development. The shift, carry, count, and complement operations, under closely controlled timing conditions, were felt to be representative of all of the operations in the manipulative elements of the type of computer planned. Accordingly, an 8-bit system using 600 transistors was designed and completed in August, 1955 and has been in nearly continuous operation since. Operating margins are periodically checked, and in steady state operation, the multiplier's error-rate has been about one every two months, or one error per 5×10^{11} multiplications at 10^5 multiplications per second. Most of these errors appear to have been caused by cracks in the printed wiring which open intermittently.

During this period, a better idea of the general characteristics of the projected computer began to develop and the engineers who were designing the 256×256 memory were encouraged to think in terms of a word of 36 bits. The notion of a logically separate input-output processor was examined and rejected in favor of a minimum buffering scheme in which data is transferred directly to and from the central memory of the computer. The possibility was recognized of programming these transfers by means of additional program sequences and associated program counters, thus taking advantage of the extensive facilities of the central machine itself for processing input-output data.

It was realized that another development step was desirable before attempting such an elaborate 36-bit system. The 8-bit multiplier had produced a certain

* The research reported in this document was supported jointly by the Army, Navy, and Air Force under contract with Mass. Inst Tech.

† Lincoln Lab., M.I.T., Lexington, Mass.

measure of confidence and familiarity with circuits, packaging, and techniques of logical design, but there remained the problems associated with communicating with memory units and input-output equipment operating at vacuum-tube levels over relatively large distances from a central machine which operated at transistor levels. It appeared that the memory development, which had now entered the construction phase, would also benefit by a preliminary evaluation of the 256×256 array and its switching, timing, and noise problems in an operating computer of some kind, possibly with a reduced word length. It was, therefore, decided to design and build next a simple machine—in fact, the simplest reasonable machine—in order to bring about an early intermediate closure of the various efforts within the program.

After some thought about the various possible minimal machines, a design was completed in which the word length would be 18 bits—a graceful half of the projected final form. We began to refer to this computer as the TX-0 and to the projected machine as the TX-2. Because the 256×256 memory array required 16 bits for complete addressing, the single-address instruction word of the TX-0 was left with 2 bits in which to encode instructions. The particular set of instructions chosen included three which required a memory address (*add*, *store*, and *conditional jump*) and one which did not. In this last instruction, the remaining 16 bits were used to control certain necessary and useful primitive operations such as clearing and complementing the accumulator, transferring words between registers, and turning on and off input-output equipment.

The TX-0, equipped with a Flexowriter, a paper-tape reader, and a cathode-ray tube display system was completed, except for the memory, in April, 1956. Twenty planes of the 256×256 memory array were installed the following August and the TX-0, now containing about 3600 transistors and 400 vacuum tubes, began to function as a complete computer. Since that time, it has been used to run a variety of testing and demonstration programs, and a symbolic address compiler and other utility programs have been constructed and are currently in use.

Not only has the TX-0 served the evaluational purposes for which it was built, but it has also demonstrated an effectiveness as a usable computer that is somewhat surprising in view of its simplicity. Its relatively high speed of about 80,000 instructions per second and its 65,536-word memory compensate in large measure for the limitations of its instruction code and logical structure.

With the successful completion of the TX-0, the final steps in the development were undertaken in packaging, circuit refinement, and logical design of the TX-2. A great deal had been learned about the performance of the transistors and memory, the types of logical circuits which are practical, techniques of marginal checking, and the lesser system problems such as color scheme selection and the proper location of pencil sharpeners. As design work progressed, the TX-2 took form as a system of about 22,000 transistors and 600 vacuum tubes. It is an interesting fact that at each step of the development since the shift register, the number of transistors involved was about 6 times the number in the preceding step. This is graphically shown in Fig. 2. At the time of writing, approximately 16 million transistor-hours have accumulated in the shift register, multiplier, and TX-0. There have been two natural deaths and a dozen or so violent ones, primarily due to contact shorting with clip leads and probes.

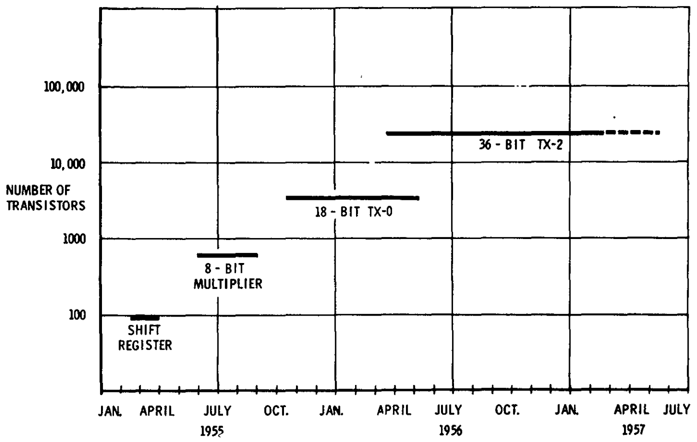


Fig. 2—Steps in the Lincoln TX-2 development program.

Design Objectives

In describing design objectives, it should be pointed out that speed of operation was *not* the primary consideration to which all other attributes were sacrificed. It would have been possible, at the expense of a few more logic circuits, to increase the speed of multiplication, division, and shift-type operations. Similarly, the operation of the index register system could have been made more efficient at the cost of an additional small, fast memory. The principal objective was rather that of achieving a balance among the factors of speed, reliability, simplicity, flexibility, and general virtue.

A key aspect is that of *expandability* which, in an experimental computer in an active environment, certainly ranks with the foregoing qualities in importance. The address structure in the TX-2 permits an expansion of the memory by about a factor of 4, partly to allow for new memory developments, such as the transistor-driven 64×64 array which was begun following the completion of TX-0. New instructions and pieces of terminal equipment will certainly be added during the course of future operation. Extra space and spare plugs have been artfully distributed about in constructing the computer frame. Finally, modular construction will permit a fairly easy physical expansion when required.

The result of all this activity has been a computer of relatively large capability. In addition to incorporating high-speed transistor circuits and a large magnetic-core memory array, the Lincoln TX-2 has two major and distinguishing design characteristics:

1) The structure of the arithmetic element can be altered under program control. Each instruction specifies a particular form of machine in which to operate, ranging from a full 36-bit computer to four 9-bit computers with many variations. Not only is such a scheme able to make more efficient use of the memory in storing data of various word lengths, but it also can be expected to result in greater over-all machine speed because of the increased parallelism of operation.

Peak operating rates must then be referred to particular configurations. For addition and multiplication, these peak rates are given in Table I.

2) Instead of one instruction counter, the TX-2 has 32 such counters which are assigned separately to different users of the computer, who then compete for operating time from instruction to instruction. A special part of the machine selects a particular user based partly on a predetermined priority schedule and partly on the current needs of that user. This multiple-sequence operation, in which many essentially independent instruction sequences interrupt and interleave one another, is an extension of the breakpoint operation found in DYSEAC of the National Bureau of Standards.

The value of these features will have to be assessed during the course of future machine operation. The features themselves are discussed in more detail in the next two papers.

TABLE I
PEAK OPERATING SPEEDS OF TX-2

Word Lengths (in bits)	Additions per second	Multiplications per second
36	150,000	80,000
18	300,000	240,000
9	600,000	600,000

Discussion

P. C. Miller (Logistics Research): When several programs are being run simultaneously, are there any provisions to prevent each of the users from stealing another's storage space?

Mr. Clark: This is a provision which, of course, has to be made by the programmer himself. Actually, suppose we were talking about the multisequence type machine that will be discussed in the next paper, I might mention that there are certain things that we can't acquire in the control of the programmer, if the programmer does try to use the same area of storage that another programmer is using, and there is no way at all that we are going to try to attempt to detect this.

F. S. Preston (Norden Labs.): What uses are planned for the TX-2?

Nelson Blachman (Sylvania EDI) and **Howard Bedford** (North American Aviation): Will the TX-2 Computer be used in any manner by the SAGE system?

Mr. Clark: What motivated the computer was an obvious desire to make something better. Is this a direct part of the SAGE development? The answer to that is, "No direct part," This is a development project of experimental systems, one of many development projects at the Lincoln Laboratory, and the Lincoln Laboratory is, of course, very deeply involved in the SAGE program, but no other form of connection exists. This machine is to be used in simulating physical systems.

W. A. Farrand (Autonetics Div. of N.A.A., Bellflower, Calif.): I would like to know what checking methods are used?

Mr. Clark: I am not sure just what checking methods are meant here. So far as the circuits go, we check the memory system by means of a parity loop; this is a very simple check. We expect the memory to be quite reliable in that we do check all of the core memories with the parity; otherwise, there is no checking while the machine is running. We have to check the machine before the machine is actually doing the programming, but this will be gone into later.

L. Kolbo (RAND Corp.): Does this machine make use of extract and deposit operations by use of masks?

Mr. Clark: The three floating functions that I mentioned originally, the and/or instructions, are not masked instructions; the only one which is, is the masked stored instruction, which is not, strictly speaking, a logical instruction but a digit and memory substitution type instruction.

W. Heising (IBM): What is the time to execute typical floating "add" (programmed)?

Mr. Clark: The reason why we did not wire floating-point operations into the machine is because we found that with configuration control it is very easy to program these instructions. Floating addition, for instance, takes 7 to 9 instructions, depending on how many there are going to be of them, to be actually executed in sequence to develop in 10 microseconds per instruction. That is, about 70 microseconds are required to execute an interpretive floating addition operation. Multiplication and division are much shorter: they take 3 or 4 instructions apiece.

R. Frohman (National Cash Register Co.): It appears that the TX-2 was well and thoroughly planned. Could you please indicate about what amount of time was spent in preliminary planning?

Mr. Clark: The preliminary planning was done largely in the previous systems which were developed. The actual manpower which went into the design and building of the computer is roughly broken down: three engineers doing logical design; three engineers doing the memory work; and three engineers designing and building the hardware. This is besides shop facilities, drafting facilities, and a good number of very good technicians. It was approximately nine people for one year.

A Functional Description of the Lincoln TX-2 Computer*

J. M. FRANKOVICH† AND H. P. PETERSON†

Introduction

THE TX-2 is a large scale digital computer designed and built at the Massachusetts Institute of Technology Lincoln Laboratory utilizing new memory and circuit components and some new logical design concepts. The computer will be applied as a research tool in scientific computations, and in data-handling and real-time problems. The design of the computer reflects not only the characteristics of the components available, but also the nature of the intended applications. This paper explains the functional and organizational aspects of the computer which are important from the user's point of view.

General Structure of TX-2

TX-2 is a parallel binary computer with a 36-digit word length. The internal memory is all random-access and will initially consist of 69,632 registers of parity checked magnetic-core memory and about 24 additional toggle switch and flip-flop registers. About 150,000 instructions can be executed per second. Instructions are of the indexed single-address type, and a fixed-point, signed-fraction, one's complement number system is used.

Several unusual ideas incorporated in the system organization reduce the amount of information unnecessarily manipulated during program sequences. Furthermore, the system organization facilitates the execution of several operations simultaneously, thereby increasing the effective speed of the computer.

The principal registers and information paths in the computer are illustrated schematically in Fig. 1. *A*, *B*, *C*, *D*, *E*, *F*, *M*, and *N* are the 36-bit flip-flop registers in the machine. *M* and *N* are memory buffer registers, each of which has a parity flip-flop and associated circuitry used to check the parity of memory words. *P*, *Q*, and *X* are 18-digit registers; *X* also has a parity digit which is used to check the parity of words in the *X* memory. Control flip-flops are not shown in Fig. 1.

Instructions are full memory words and are placed in the Control Element during the instruction memory cycle. During the operand memory cycle, an operand is usually transmitted between the Memory Element and some other element—always through the Exchange

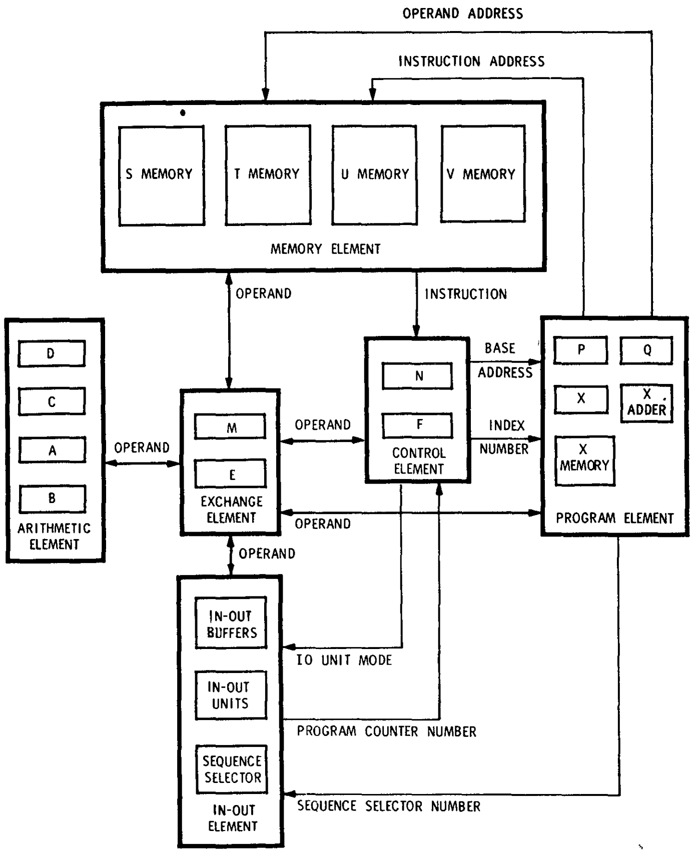


Fig. 1—TX-2 system schematic, showing the principal registers and transfer paths.

Element. The 36-digit configuration of the memory is not, however, maintained throughout the computer during operation timing. A programmer can, in effect, control several independent, shorter operand word length computers simultaneously during the execution of each instruction. This flexibility is realized by specifying a particular system *configuration* with each instruction.

The computer communicates with the outside world through units in the In-Out Element, several of which can be simultaneously operated. Whenever an input or output information transfer can occur, signals to the Program Element from the In-Out Element automatically call into operation the associated instruction sequence. This *multiple-sequencing* aspect of the computer will not be described in this paper.[1]

* This research was supported jointly by the Army, Navy, and Air Force under contract with the Mass. Inst. Tech., Cambridge, Mass.

† M.I.T. Lincoln Lab., Lexington, Mass.

[1] J. W. Forgie, "The Lincoln TX-2 in-out system," this issue, p. 156.

Memory Element

The availability of a large, fast, core memory for TX-2 permitted an emphasis on the design of a machine with an all random-access memory which could be as large as 262,144 words. The homogeneous aspect of so large a memory system simplifies the programmer's coding problems and permits continued high-speed operation regardless of the program location in the internal memory.

The TX-2 Memory Element (see Fig. 2) is divided into four independently operating memories, each containing up to 65,536 36-digit words. The operating speed of TX-2 is determined by the cycle time for the memories: the 65,536-word *S Memory* is expected to have a cycle time of between six and seven microseconds; and the 4096-word *T Memory*, a cycle time between five and six microseconds. Both memories are parity checked.

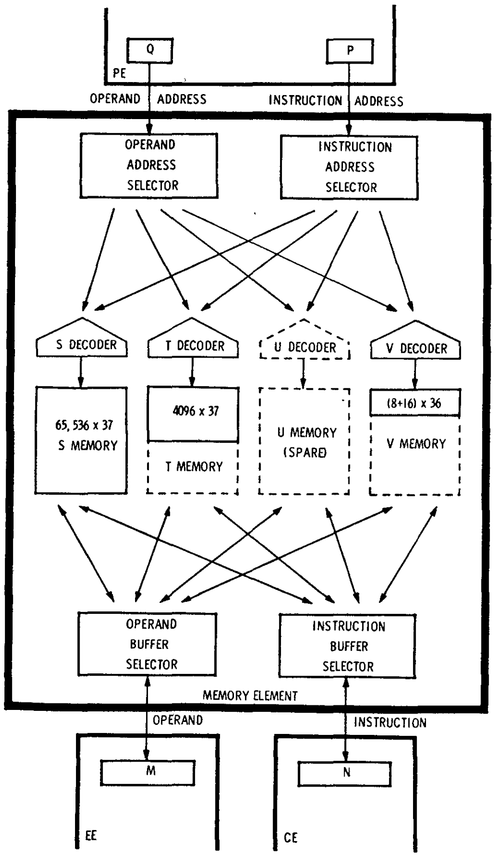


Fig. 2—TX-2 Memory Element. Two address and two buffer registers are used to permit simultaneous operation of any two of the four memories.

Although the *U Memory* currently is not specified, it may contain a 4096-word core memory in the initial system. The *V Memory* consists of 8 flip-flop registers in the central machine and 16 toggle switch registers which contain the program sequence executed whenever the START button on the operator's console is pushed. The contents of the toggle switch registers can be used as instructions or operands, but naturally cannot be altered by a program. The six 36-bit registers *A*, *B*, *C*, *D*, *E*, and *F* are also part of the *V* Memory but their contents can be used only as operands during the execution of an instruction. The programmer has, in a limited sense, a two address instruction machine when he refers to these registers in load and store type instructions. The other two flip-flop registers in the *V* Memory are a 60-counts-per-second clock and a random-number register.

When an instruction calls for the storing of an operand in memory, the operand memory cycle can be extended up to two microseconds. The extension occurs between the time that the memory register is read and the time that it is rewritten. During this extension time the memory register transfers in the central computer take place, the parity of the word read from memory is checked, and the parity of the new memory word computed. Because the extended cycle is less than the two complete cycles traditionally used for word-modifying instructions, an increase in computing efficiency is realized.

The *P* Register in the Program Element specifies the location of an instruction in memory and the *N* Register in the Control Element holds the instruction after it has been read from memory. The two leftmost digits of *P* select the memory system from which the instruction word is to be obtained; the right 16 digits address the word within the memory. Similarly, the *Q* Register locates the operand in one of the memory systems, the operand being placed in *M*.

Control and Indexing

An instruction word read into *N* has the structure shown in Fig. 3. The first two digits of the word specify information to the In-Out Element, and the four *cf* digits specify the computer configuration. The interpretation of the *b* and *d* digits is not discussed here.[2] The *cf* digits will be discussed later.

The operation code for the instruction is specified by

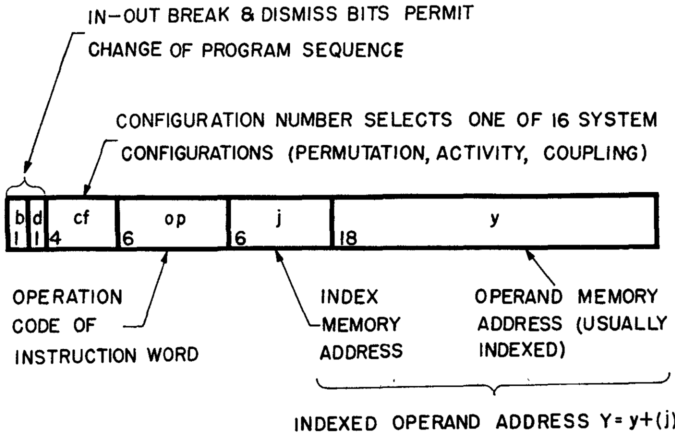

IN-OUT BREAK & DISMISS BITS PERMIT CHANGE OF PROGRAM SEQUENCE
CONFIGURATION NUMBER SELECTS ONE OF 16 SYSTEM CONFIGURATIONS (PERMUTATION, ACTIVITY, COUPLING)
b 1
d 1
cf 4
op 6
j 6
y 18
OPERATION CODE OF INSTRUCTION WORD
INDEX MEMORY ADDRESS
OPERAND MEMORY ADDRESS (USUALLY INDEXED)
INDEXED OPERAND ADDRESS Y = y+(j)

Fig. 3—TX-2 instruction word layout.

[2] *Ibid.*

the six *op* digits. On simple load and store type instructions these six digits are further subdivided into two groups of three. The first group determines the operation and the second specifies the register in the central computer which is being loaded or whose contents are being stored.

The base address for the operand, formed by the 18 *y* digits, is usually modified by the contents of the index register selected by the six *j* digits. The index registers form a unique 64-register, parity-checked core memory which has a 1 microsecond access time. The contents of the specified index register is read into the *X* Register of the Program Element via the paths indicated in Fig. 4. The base address and the index are fed into a full adder circuit which produces the sum, $Y=y+(j)$, in about 1 microsecond. The over-all complexity of the Program Element was reduced by having the adder produce both the sum, Y, and the unmodified base address, y; either of these quantities can be directed to the operand memory address register Q. Whenever the zeroth index register is chosen, the adder produces only the unmodified base address. The effect is the same as having the index register contain zero, so the programmer can avoid index modification altogether.

The instruction memory address register P normally is indexed by one as each instruction is executed, but jump instructions may cause the output of the index adder to be directed to P. The adder also provides a communication path for index jump instructions from the X Memory to the Memory Element by way of the Exchange Element.

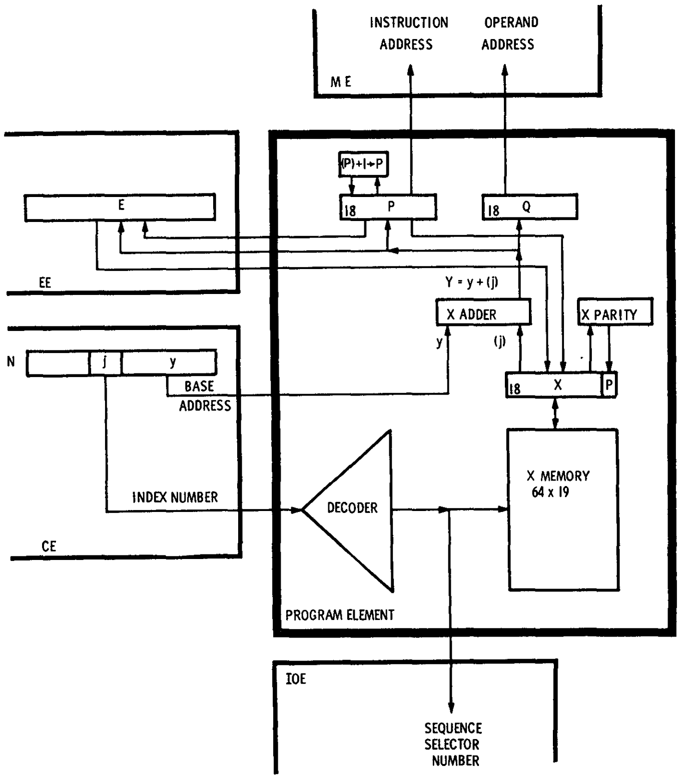


Fig. 4—TX-2 Program Element, determining the instruction and operand memory addresses, performing X memory operations.

Arithmetic Element

The registers and sufficient basic operations in the Arithmetic Element (AE) to implement addition, multiplication, division, shift, and various logical operations are shown in Fig. 5. Operation timing for most of the TX-2 instructions is also performed in the AE.

The design of the AE reflects the desire to attain high-speed operation for TX-2 even when long-time instructions are being performed in the AE. The only instructions which require more than a memory-cycle time for execution are those which involve shifting. These are, for example, multiply, divide, shift, and normalize. For this reason the AE contains a sufficient number of storage registers to permit these instructions to be carried out in the AE while the remainder of TX-2 is freed to perform other instructions.

The four registers in the AE can each communicate with the E Register in the Exchange Element and thus with the Memory Element. As mentioned earlier, these registers are addressable as part of the V Memory System. Therefore, programmers have access to the results in any register of an AE computation.

The AE registers, designated by A, B, C, and D, are described on the next page.

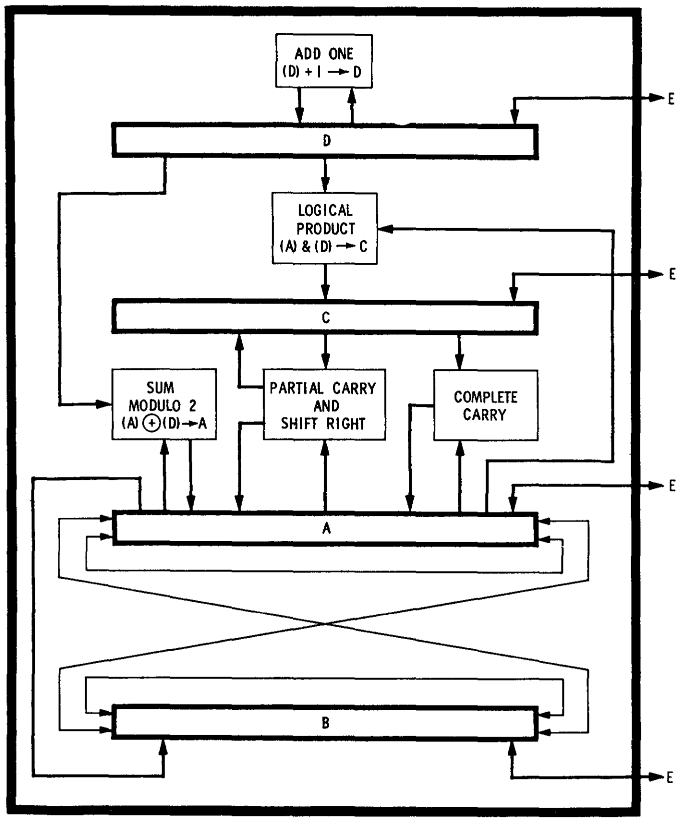

ADD ONE (D) + 1 → D
E
D
LOGICAL PRODUCT (A) & (D) → C
E
C
SUM MODULO 2 (A) ⊕ (D) → A
PARTIAL CARRY AND SHIFT RIGHT
COMPLETE CARRY
E
A
B
E

Fig. 5—TX-2 Arithmetic Element, showing the circuits and transfer paths for AE operations.

The A Register accumulates the results of all the arithmetic operations except division for which it holds the remainder. It holds one of the operands and accumulates the results of the three logical operations (AND, INCLUSIVE OR, EXCLUSIVE OR) which, it should be noted, are bit-wise operations. The information in the A Register can also be shifted (*i.e.*, multiplied by some positive or negative power of two) or cycled (*i.e.*, shifted, without preserving the special significance of the sign bit, as in a closed ring).

The B Register serves as an extension of A during multiplication, certain shifts and cycles, and, in a sense, during division when the least significant digits of the double-length dividend are stored in B. The resulting quotient then appears in B. Moreover, the information in B can be shifted or cycled independently of A. In multiplication, the multiplier originally in A is transferred via parallel paths directly into B (where the least significant digit then controls the operation).

The C Register stores the partial carries during arithmetic operations, most important during multiplication as described later. Since these partial carries are actually bit-wise logical products (AND), C is also used to accumulate logical products.

The D Register holds the multiplicands, divisors, addends, and one of the operands for the logical operations. It also holds the numbers which control the shifting and cycling of A and B, namely the number of places, up to 72, and the direction, right or left. The facility of D to count is used also in accumulating the results of the normalizing of A and counting ones in A.

Besides the above mentioned facilities, each of the AE registers can be complemented, which allows subtractions to be done.

AE Circuits

There are four *Add One* circuits on D, so that different parts of A and B can be controlled separately and simultaneously. For simplicity, just one *Add One* circuit is shown in Fig. 5. These *Add One* circuits use the simultaneous carry principle, permitting one count to occur every 0.4 microsecond. Each can count up to 127.

The *Logical Product* circuit of A and D into C and the *Sum Modulo 2* (EXCLUSIVE OR) circuit of A and D into A when used at the same time are called a *Partial Add.* When the *Complete Carry* circuit is activated after a Partial Add, the result is a full addition of D and A into A. The *Complete Carry* circuit uses the high-speed carry principle and takes about 1.5 microseconds for 36 bits.

The *Partial Carry and Shift Right* circuit is also known as "multiply step" and was, we believe, first used on Whirlwind I. As used in multiplication, this circuit obviates the need for a full addition for each "one" in the multiplier. Carries are propagated only one stage during each step except the last when a complete carry is executed. This iterative process takes about 16 microseconds in the worst case for a full 36-digit multiplication. The iterative process for division, on the other hand, requires a complete addition at each step and consequently takes about 72 microseconds in the worst case.

Two features of the AE control ought to be mentioned here. A 7-bit step counter, like the Add One circuit on D, is used to control multiplication and division and to limit the shifting in normalizing and the cycling in counting "ones." A flip-flop signifying overflow during addition and division is also used to remember the sign of the product during multiplication and the sign of the quotient during division. If a division overflow occurs, the sign is replaced by the overflow state and the quotient is lost.

Control of the Arithmetic Element is independent of the rest of the machine, thus providing the time-saving device of continuing to execute non-AE instructions while AE is performing one of the longer shift operations or a division.

System Timing

In part, the high speed of TX-2 is attained by overlapping the operation of as many components as is logically possible without incorporating large amounts of circuitry. The time-consuming cyclic operations in an indexed single-address computer are the *instruction-memory cycle*, the *index-memory cycle*, the *index-addition time*, the *operand-memory cycle*, and the *operation timing*. These cycles occur in the mentioned sequence during the execution of ordinary instructions.

Several asynchronous "clocks" which use a 5-megacycle pulse source control the various cycles. The instruction and operand memory cycles can be overlapped if they take place in different memory systems.

The overlap of these cycle times for a sequence of load type instructions is illustrated in Fig. 6(a). Here different instruction and operand memories with roughly equal cycle times are assumed. If a sequence of store type instructions is executed which requires extended memory cycles for the operand, then the situation shown in Fig. 6(b) results. Fig. 6(c) shows the time used when both the instruction and the operand are in the same memory.

"Peak" operating speed for the computer is attained only in Fig. 6(a); additional circuitry could improve Fig. 6(b) and Fig. 6(c), but only at considerable cost. It is interesting to note that if the computer is to run at peak speeds, the address of the operand used by the current instruction must be available before the earliest moment at which the next instruction memory cycle could begin. If the total accumulated time from the beginning of an instruction memory cycle until the time that the address of the operand is known is greater than the instruction memory cycle time, then the computer

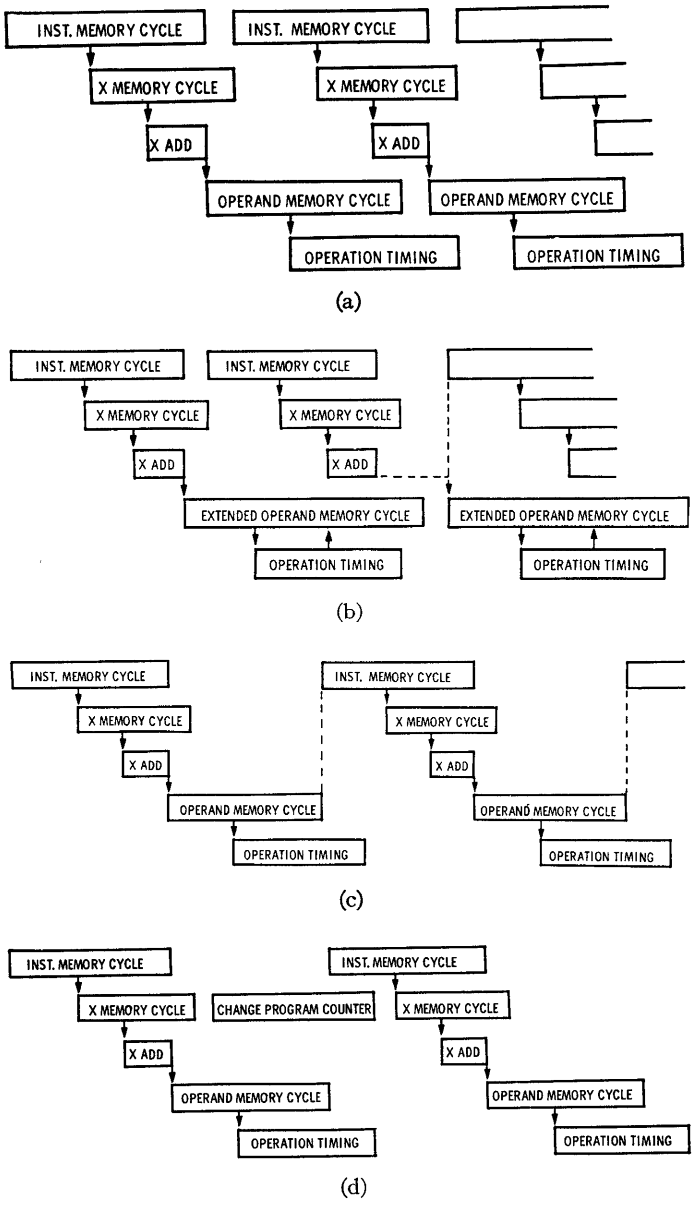


Fig. 6—TX-2 timing schematic, showing overlapped execution of memory and operation cycles. (a) Consecutive load type instructions—instructions and operands in different memories. (b) Consecutive store type instructions. (c) Instruction and operand in same memory. (d) Change sequence.

cannot run in the ideal manner shown in Fig. 6(a). This means that the access time of all memories, and the index add time must be kept as short as possible.

Fig. 6(d) depicts the timing of events when the In-Out Element causes a change in program sequence by changing the contents of the P register. The additional X Memory cycles which must be performed in carrying this out produce a timing situation similar to that of the X Memory load and store type instructions.

The operation timing for an instruction is executed when the operand is available from memory. Only the Arithmetic Element step counter instructions, multiply, divide, shift, etc., require an operating timing cycle longer than a memory cycle. Since only the Arithmetic Element is tied up when these instructions occur, the Control Element permits any non-Arithmetic-Element instruction to be executed while the AE is busy. Division takes up to 75 microseconds, so the programmer can write as many as 14 non-AE instructions following a divide, all of which can be executed before the division is completed.

Configuration

The design of a general purpose computer must necessarily reflect the contradictory demands for both short and long word lengths, floating and fixed point arithmetic operations, and a multitude of logical and decision instructions. The computer should be able to process information at an optimum rate in a variety of problems without the need for intricately coded programs. This ability should be achieved without excessively complex and costly circuitry.

The full 36-digit word in TX-2 represents a reasonable length for operands in some numerical computations, notably scientific and engineering computations. Though floating point arithmetic operations are not included in the instruction code, both they and multiple-precision operations can be easily synthesized by means of the existing instructions. The logical instructions in the code facilitate operations on individual digits, but also, a configuration which the programmer specifies anew with each instruction permits him to perform arithmetic operations on operands which are less than 36 digits long. When such is the case, several shorter operands can be manipulated simultaneously.

The four *cf* digits in an instruction word (see Fig. 3) are decoded as shown schematically in Fig. 7. The contents of the selected one of 16 9-digit configuration words are placed in a flip-flop register whose output levels determine a static configuration for the entire computer during the execution of the instruction. The contents of the first twelve registers are specified by a notation whose meaning will be clarified in the following discussion.

The full 36-digit word length is always maintained for instruction words, but during operation timing, every 36-digit register in the Memory, Exchange, and Arithmetic Elements is considered broken into four 9-digit *quarters* [numbered from 1 to 4, from right to left as in Fig. 8(a)]. While the instruction is being executed, these quarters are recombined on the basis of the configuration.

Parallel register transfers are the usual means for moving information about in the machine. The *EE permutation* digits select one of the four permutations P0, P1, P2, or P3 as defined in Fig. 8(b). The chosen permutation effects the corresponding cross-communication paths between the quarters of the E and M registers of the Exchange Element. As operands are transmitted through the EE, the quarters of the word follow the set of paths determined by the selected permutation. The result is that the operand is shifted $9n$ places to the left as it moves from M to E or $9n$ places

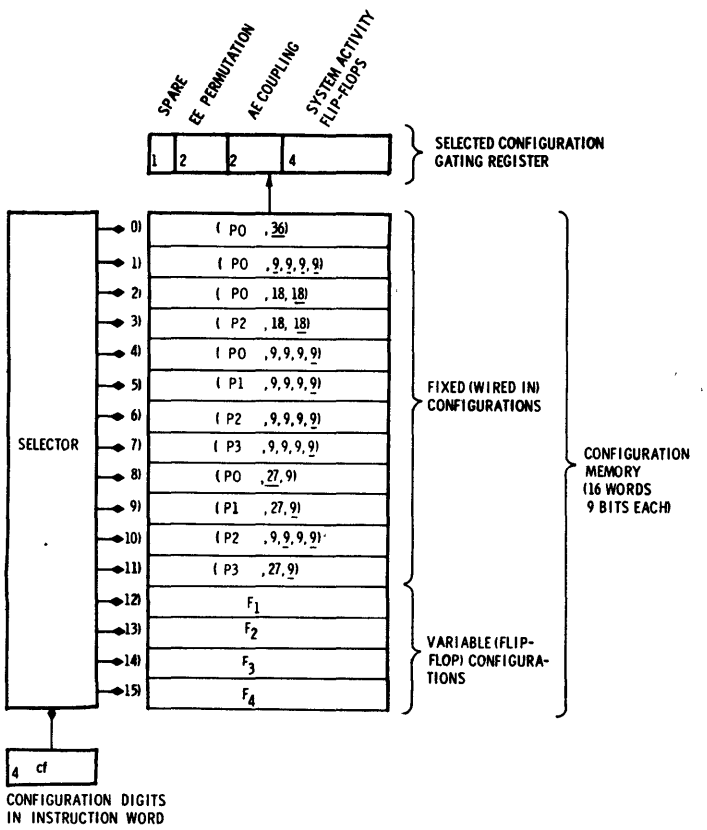


Fig. 7—TX-2 configuration selection. The *cf* digits select a configuration for the computer for use during the execution of the instruction.

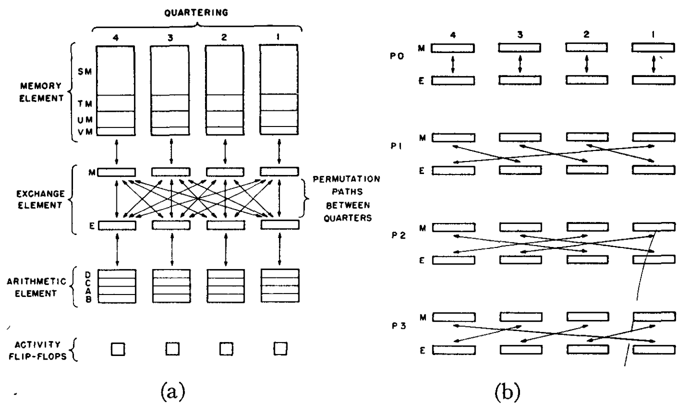


Fig. 8—TX-2 configuration. (a) Quartering, permutation paths, and activity flip-flops. (b) The four sets of permutation paths available, one of which is used during the execution of an instruction.

to the right as it moves from E to M, $n=0$, 1, 2, or 3. Thus the programmer can have any quarter of the AE communicate with any quarter of the ME.

This communication ability is focused more sharply by having the configuration specify a *system activity*. All operation timing events in a given quarter of the AE and EE and the quarter of the ME connected via the selected permutation path in the EE are controlled by the activity flip-flop of that quarter. If the activity flip-flop of a given quarter holds a "one," as specified by the configuration, then the operation timing events of the instruction occur in that quarter. If the activity flip-flop holds a "zero," then nothing happens.

During the execution of arithmetic operations, the *AE coupling* bits further specify the connections of the lateral information paths between quarters in the AE. Information flows laterally only through the shift and the carry circuits, and the connection of these circuits alone determines the word length of the numerical quantities manipulated in the AE.

In Fig. 9(a) (next page) every quarter of the AE has coupling units at each end which receive the shift and carry information entering the quarter. The general type of connections among several quarters is shown in Fig. 9(b). The digit *length* of operands during add and shift operations is determined by the number of quarters coupled together. In TX-2 from one to four quarters can be coupled together to permit arithmetic operations on 9, 18, 27, or 36-digit operands. The various combinations of coupling unit connections actually chosen by the AE coupling are symbolized in Fig. 9(c). Since *A*-register, *B*-register, and *AB*-register shifts are permitted in the Arithmetic Element, the programmer can obtain 18, 36, 54, or 72-digit shifts. All the possible shift (and cycle) configurations are shown in Fig. 9(d).

Only those inputs to the coupling units which would yield useful arithmetic element structures are realized by the AE coupling. It should be emphasized that the programmer can realize several arithmetic elements simultaneously. The coupling (36) gives only one 36-bit AE, but the coupling (18, 18) gives two complete, independent 18-bit arithmetic elements which are separately but simultaneously controlled by the instruction being executed. Two arithmetic elements are again available with the coupling (27, 9), one 27 bits and the other 9 bits long, and the (9, 9, 9, 9) case gives four 9-bit arithmetic elements. The permutation paths in the Exchange Element permit each arithmetic element to communicate with any quarter of a memory word and the activity flip-flops can specify just which of the realized arithmetic elements will actually be active and in active communication with the connected part of memory.

In Fig. 10, several examples are given of the different configurations which can be realized in TX-2. The most straightforward configuration has one 36-digit arithmetic element and communicates directly with memory. The notation (P0, $\underline{36}$) signifies the permutation (no shift) and the form of the arithmetic element (one 36-digit). The underlining indicates that the whole system is active. Slightly more varied is the (P0, 9, $\underline{9}$, 9, $\underline{9}$) configuration which specifies four 9-digit arithmetic elements communicating directly with memory, but with only two of them active. The (P2, 9, $\underline{9}$, 9, $\underline{9}$) configuration has the same arithmetic elements but with the associated memories interchanged. The (P2, 18, $\underline{18}$) configuration illustrates an 18-digit arithmetic element which uses the "other" half of memory.

One of the 9 configuration digits is at the moment unused, but will probably be used to control the ex-

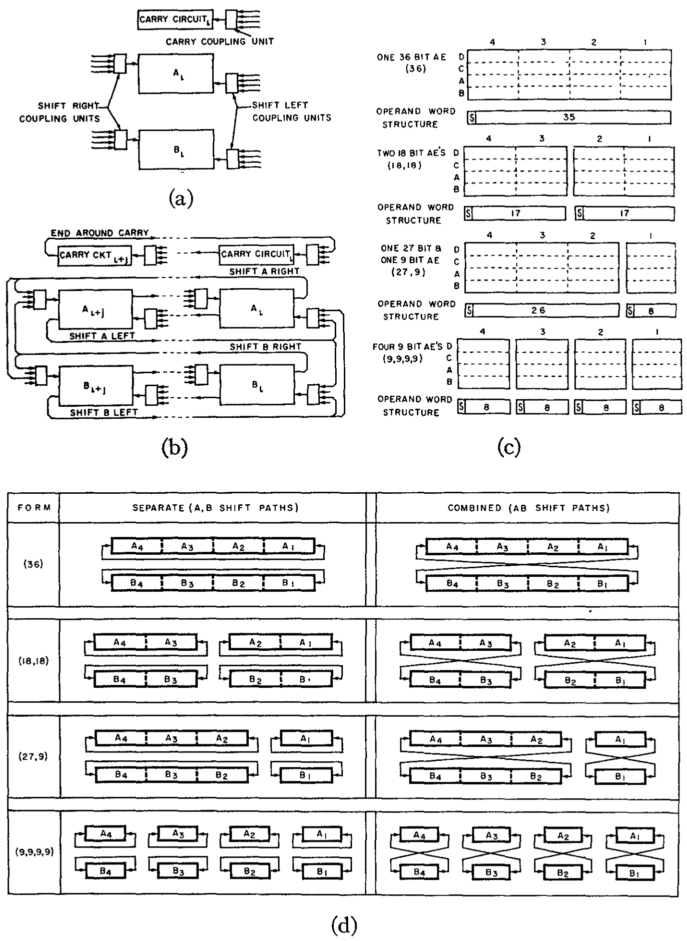


Fig. 9—TX-2 arithmetic element coupling units. (a) i-th quarter coupling units. The coupling units receive information moving laterally into the i-th quarter of the AE, $i=1, 2, 3, 4$. (b) Coupling unit connections between a contiguous group of quarters which realize a 9-bit ($j=0$), 18-bit ($j=1$), 27-bit ($j=2$) or 36-bit ($j=3$) "arithmetic element." (c) Arithmetic element and operand word structures. The four forms the arithmetic element can assume with associated operand word structure. (d) The possible shift path arrangements realized with the configurations.

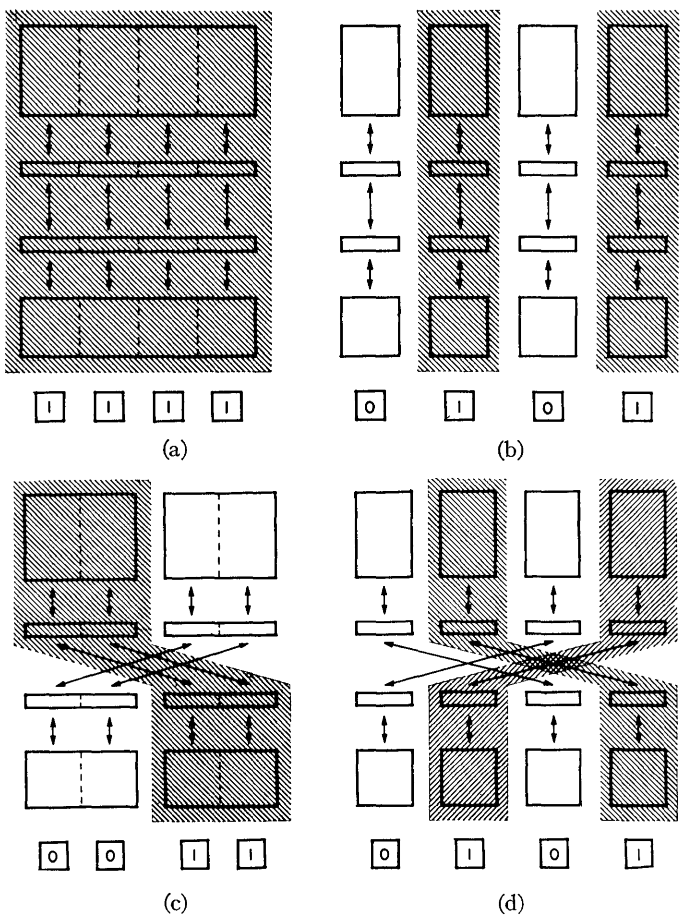


Fig. 10—Illustrative example of different configurations. Areas of activity during execution of instruction are shown shaded. Effects of AE coupling are shown by juxtaposition. (a) (P0, $\underline{36}$) configuration. (b) (P0, 9, $\underline{9}$, 9, $\underline{9}$) configuration. (c) (P2, 18, $\underline{18}$) configuration. (d) (P2, 9, $\underline{9}$, 9, $\underline{9}$) configuration.

tension of the sign of numbers as they pass through the EE on the way from the ME to the AE. The scheme presently under consideration would permit programmers to add, for example, a 9-digit memory operand to an 18-digit arithmetic element. This scheme would permit closer packing of operands in memory and significantly increase the speed of solving some real-time problems, where short data words need to be extended so higher precision can be maintained during computations. Working details of the scheme have yet to be fixed.

The configuration memory from which the programmer chooses a configuration for use with each instruction was shown in Fig. 7. Twelve of the configuration memory registers are fixed circuitry whose contents cannot be changed without changing the wiring of the computer. These configurations are assumed to be ones which will be useful to most programmers. The last four registers in the memory consist of the 36 digits of the F register. As will be seen the programmer can quite simply alter the contents of this register and thereby obtain any of the (less than 2^9) possible configurations.

Instruction Code

Of the 64 possible operation codes, only 51 are currently decoded to define instructions. In Table I (opposite) the effect of each instruction is described. If several computers are defined by the configuration, then the effect occurs in all of them simultaneously and independently. The notation used in the definition of the operation is described in Table II (p. 154).

The instructions are grouped according to type. Load and store type instructions simply effect an operand transfer between the selected register and memory. The load complement instructions are variants which load the one's complement into the specified registers. Exchange simply interchanges the contents of A and the indicated memory register. The insert instruction allows any set of bits in A, as specified by the bits in B, to be stored in memory. In the index memory load and store instructions, the j bits select the index register involved so the operand address is not modified.

All of the add and step-counter instructions can also be classed as load type instructions in so far as the operand memory cycle is concerned. The multiply instruction forms the full product in the A and B registers. Division is the inverse of multiplication, the double

TABLE I

Type	Mnemonic Code	Operation	Name
Load	lda	$(\bar{Y}) \to \left\{\begin{array}{l} A \\ B \\ C \\ D \\ E \\ F \end{array}\right.$	Load into *A*
	ldb		Load into *B*
	ldc		Load into *C*
	ldd		Load into *D*
	lde		Load into *E*
	ldf		Load into *F*
	lca	$\overline{(Y)} \to \left\{\begin{array}{l} A \\ B \end{array}\right.$	Load complement into *A*
	lcb		Load complement into *B*
	ldx	$(y) \to j$	Load into index
Store	sta	$\left.\begin{array}{l} (A) \\ (B) \\ (C) \\ (D) \\ (E) \\ (F) \end{array}\right\} \to Y$	Store *A*
	stb		Store *B*
	stc		Store *C*
	std		Store *D*
	ste		Store *E*
	stf		Store *F*
	exa	$\left\{\begin{array}{l} (Y) \to \mathrm{A} \\ (A) \to Y \end{array}\right\}$	Exchange *A*
	ins	$(B)\ \&\ (A) \vee \overline{(B)}\ \&\ (Y) \to Y$	Insert digits of *A*
	stx	$(j) \to y$	Store index
Add	add	$(A) + (Y) \to A$	Add
	sub	$(A) + \overline{(Y)} \to A$	Subtract
	dma	$\lvert (A) \rvert + \lvert \overline{(Y)} \rvert \to A$	Difference of magnitude
	and	$(A)\ \&\ (Y) \to A$	Logical and
	ori	$(A) \vee (Y) \to A$	Logical or—inclusive
	ore	$\left\{\begin{array}{l} (A) \oplus (Y) \to A \\ (A)\ \&\ (Y) \vee (C) \to C \end{array}\right\}$	Logical or—exclusive (and accumulate product)
	axm	$(j) + (y) \to y$	Add index to memory
	amx	$(j) + (y) \to j$	Add memory to index
Set bit	sbo	$1 \to Y_j$	Set *j*-th bit one
	sbz	$0 \to Y_j$	Set *j*-th bit zero
	sbc	$\overline{(Y_j)} \to Y_j$	Set *j*-th bit complement
Step-Count	mul	$(A) \times (Y) \to AB$	Multiply
	div	$(AB) \div (Y) \to \left\{\begin{array}{l} A \text{ (remainder)} \\ B \text{ (quotient)} \end{array}\right.$	Divide
	sha	$\left.\begin{array}{l} (A) \\ (AB) \\ (B) \end{array}\right\} \times 2^{(Y)} \to \left\{\begin{array}{l} A \\ AB \\ B \end{array}\right.$	Shift *A*
	sab		Shift *AB* together
	shb		Shift *B*
	cya	$\left.\begin{array}{l} (A) \\ (AB) \\ (B) \end{array}\right\} \mathrm{cyc}(Y) \to \left\{\begin{array}{l} A \\ AB \\ B \end{array}\right.$	Cycle *A*
	cab		Cycle *AB* together
	cyb		Cycle *B*
	nab	$\left\{\begin{array}{l} (AB) \times 2^{nf} \to AB \\ (Y) - nf \to D \end{array}\right.$	Normalize *AB*
	coa	$(Y) + no \to D$	Count ones in *A*
In-out	rds	$\left\{\begin{array}{l} (Y) \to IO \\ (IO \to Y \end{array}\right\}$	Read and shift
	rdn	$\left\{\begin{array}{l} (Y) \to IO \\ (IO \to Y \end{array}\right\}$	Read without shift
Jump	jpe	If $(E_j) = 1$, then $y \to P$	Jump if *j*-th bit of *E* is a one
	jpp	$\left.\begin{array}{l} \text{If any } (A) > 0 \\ \text{If any } (A) \leq 0 \\ \text{If any } (A) = 0 \\ \text{If any } (A) \\ \quad \text{overflowed} \end{array}\right\} \text{then } Y \to P$	Jump if the contents of any *A* is positive
	jpn		Jump if the contents of any *A* is negative
	jpz		Jump if the contents of any *A* is zero
	jpo		Jump if the contents of any *A* has overflowed
	jxp	If $(j) \geq 0$, then $(j) - cf \to j,\ y \to P$	Jump if index positive and decrease index
	jxn	If $(j) < 0$, then $(j) + cf \to j,\ y \to P$	Jump if index negative and increase index
	jpu	$\left\{\begin{array}{l} \text{If } cf = 1, 3, \text{ then } (P) + 1 \to j \\ \text{If } cf = 0, 1, \text{ then } y \to P \\ \text{If } cf = 2, 3, \text{ then } Y \to P \end{array}\right\}$	Jump unconditionally
Misc.	ios		In-out select
	opr		Operate

length dividend in *A* and *B* being divided by the memory operand. The remainder is left in *A* and the quotient in *B*. Normalize shifts the contents of *A* and *B* left until the magnitude of the number in *A* is between one-half and one. The number of shifts to do this, the normalizing factor, is subtracted from the memory operand in *D*. The shift and cycle instructions use the memory operand, rather than the address section of the instruction, to specify the number of places to shift. This is necessary since more than 18 bits are required to specify all the possible shifts for the ($\underline{9}$, $\underline{9}$, $\underline{9}$, $\underline{9}$) configuration. The count ones instruction adds the number of bits in *A*

TABLE II

Notation	Meaning
$\rightarrow$	goes into
(x)	contents of x
$Y=y+(j)$	indexed memory address
$\lvert(x)\rvert$	magnitude of (x)
$\overline{(\bar{x})}$	one's complement of (x)
&	logical and operation
$\vee$	inclusive or operation
$\oplus$	exclusive or operation
$+$	one's complement addition
nf	number of shifts to normalize
no	number of ones
Y_j	j-th digit of register Y

which are ones to the memory operand in D. This provides a simple means for determining bit density in areas of storage, since the one's count for several words can be accumulated in D.

The two replace add instructions, using the index memory, facilitate instruction and index modification. Both require two memory cycle times for execution.

The two in-out read instructions transmit information between the memory and the selected in-out unit. The details of these and the in-out select instruction are given in another paper.

Single bits in memory can be manipulated with the three bit-setting instructions. The bit-sensing instruction facilitates the use of single bits in memory as operands.

The variety of jump instructions available simplifies the coding of logical decision functions. The two-index jump instructions permit indexed program loops to refer successively in either the forwards or backwards direction to operands in a data block. The unconditional jump instruction uses the *cf* digits to specify whether the selected index register will be used to remember the previous contents of P. These contents are always transmitted to the E register whenever a jump occurs.

Arithmetic overflows can be caused by addition, subtraction, and division instructions. Such overflows as do occur are remembered in overflow flip-flops in the arithmetic element. The overflow condition can be detected by a jump instruction, or by the in-out element in a manner described in another paper. If an overflow is anticipated, however, it can be shifted into the A register by executing a normalize instruction. A normalize usually shifts AB left, but if an overflow exists AB is shifted right one place, and the overflow placed in the most significant digit position of A to the right of the sign digit. The memory operand is increased by one in the D register, when this occurs, rather than decreased. This interpretation of an overflow permits floating-point operations to be programmed quite simply in the arithmetic element. The in-out select and operate instructions differ from all the others in the sense that the y digits are used to specify different operations. In-out select chooses the mode in which an in-out unit will run. The operate instruction will control individual useful commands, as for example, round-off.

Instruction Times

The average execution time for instructions depends upon whether one memory or two different overlapped memories are used for instructions and operands. In the latter case the average time is the longer of the instruction memory and the operand memory cycle times, and in the first case the sum of the two cycle times. It should be remembered that any instruction which involves storing an operand in memory has the normal operand memory cycle time extended by from one to two microseconds. Instructions which alter or transfer the contents of index memory registers, require approximately two normal memory cycles even when instruction and operand memory cycles are overlapped.

Successive step counter instructions require a time which depends upon the length of the longest active arithmetic element. In the case of multiply, divide, and count ones, this time is a function of the operand word length only, but the shift, cycle, and normalize times depend upon the number of places actually shifted. Divide requires about 2 microseconds per digit and all other step counter instructions 0.4 microsecond per digit. These shift times become significant only when they exceed the one or two memory cycles already required. In the worst 36-digit case about 75 microseconds is required for division and 19 microseconds for multiplication. A 72 place shift would take 32 microseconds. These are the times required for these instructions when they are written in sequence. If the operand word length is shorter, then these times become proportionally less, down to the minimum memory times required.

Conclusion

The organization of TX-2 permits a programmer to pay considerable attention to coding details and receive a worthwhile reward in the form of increased efficiency of operation. The operating speed can be doubled when instructions and operands are stored in different memories. Further increases result by the sequencing of instructions so that non-Arithmetic-Element instructions are executed concurrently with AE step-counter instructions. And the ability to choose a configuration with each instruction means not only that some instructions take less time, but also that many of them can be eliminated from a program altogether.

However, this versatility and efficiency is not accompanied by a disastrous loss in simplicity. The system organization is such that details can be easily ignored by the naive programmer, without the details having even subtly obtrusive effects. If all the digits in an instruction word are zero except for the operation code and the base address, then TX-2 appears as a simple single address

36-bit operand word computer with a single, uniformly addressed 70,000 word memory.

If the *j* bits are used, then the machine is enlarged to become an indexed single-address 36-bit operand word computer for which the entire instruction code is meaningful. When the *b* and *d* bits are used, then the programmer can control the manner in which several in-out units running concurrently can cause program sequence changes. And by selecting various configurations the programmer can perform more operations simultaneously with each instruction.

The different facilities for indexing, memory overlap, instruction overlap, multiple-sequencing, and configuration can be ignored or used as the programmer desires. Ignoring them would seem to permit straightforward coding; using them actually permits much shorter and faster codes for a given function. Each facility is easily represented by a clear conceptual picture of what the facility permits, the only real difficulty being the greater number of simultaneous actions possible with each instruction. However, higher speeds and greater system capacity are obtained by shorter cycle times, increased bit storage, and greater simultaneity of events. In TX-2 all three aspects are emphasized.

Discussion

C. H. Richards (Convair-Astronautics): What is the accumulator length of TX-2, and where is the binary point located?

Mr. Frankovich: This is a 36-bit word accumulator, in a ones-complement machine. The binary point really exists only by virtue of what happens during multiplication or division type instructions. The left digit is the sign digit of whatever configuration you have, and the remaining digit is a numeric digit; and ordinarily during multiplication you can consider the binary point to be between the sign digit and the first significant digit on the remainder of the operand. During division, however, we have a different interpretation, so we cannot really say that this is a fractional machine. During addition it makes no difference where you put the binary point. During division the quotient is generated in a different register than the accumulator, so we cannot say that it is a fractional machine during that operation.

Chairman Pfister: When you multiply two 36-bit words, together you have a 72-bit product; where does the product go when you have an accumulator with only 36 bits?

Mr. Frankovich: There are 4 registers in the arithmetical element; and another register which acts as the right-hand extension of this accumulator—a B register. During multiplication the full 72-digit product is generated in the accumulator in the B register. The binary point is at the left-hand of the accumulator during the entire process. The other two registers are used, one to hold the partial carry during addition operations; another is used to carry out division, thereby enabling the arithmetical element to be completely selfcontained during such a long period of instruction.

D. L. Shell (General Electric): What happens on overflow?

Mr. Frankovich: We have four overflow indicators in the arithmetical element. If we have a full 36-bit operand for an instruction, then we use only the left-most overflow indicator, and associate one overflow indicator with each quarter; none of the other overflow indicators are affected at all. On the other hand, if we have four 9-bit operands, then we use all four overflow indicators to indicate overflow for any one of them.

I might also mention that during the jump on overflow instruction you can specify it by means of configuration control, in a very straightforward manner. There are further techniques for handling such situations which are devised to make programming easier.

Mr. Groelinger (Ramo-Wooldridge): Can the exchange element be used to store accumulator content in *several* places in memory?

G. G. Chapin (Remington Rand UNIVAC): Can you read from one memory element into more than one arithmetical element?

Mr. Frankovich: This can be done in two ways. As far as the one and one instruction in each transfer: if you want to store one-half of the arithmetical element in several places in the memory, be it in the left half, or the right half, wherever your location might be, then you give instructions to each transfer, unless the transfer were to be done in the same register. If you are loading the arithmetic element, you can load either half of the accumulator from a given memory register, but again this takes two instructions.

G. G. Chapin (Remington Rand UNIVAC): Can jump instructions be conditioned on more than one 9-bit section of the accumulator simultaneously?

Mr. Frankovich: Yes. The configuration control device is used universally and homogeneously upon all arithmetical instructions. If you have four 9-bit operands, and you want to jump on the basis of two of them, the jump instruction is interpreted to be "jump on either the first-quarter or the third-quarter."

The Lincoln TX-2 Input-Output System*

JAMES W. FORGIE†

Introduction

THE input-output system of the Lincoln TX-2 computer contains a variety of input-output devices suitable for general research and control applications. The system is designed in such a way that several input-output devices may be operated simultaneously. Since the computer is experimental in nature, and changes in the complement of input-output devices are anticipated, the modular scheme used will facilitate expansion and modification. The experimental nature of the computer also requires that the input-output system provide a maximum of flexibility in operating and programming for its input-output devices.

The input-output devices, currently scheduled for connection to TX-2, include magnetic-tape units for auxiliary storage; photoelectric paper-tape readers for program input; a high-speed printer, cathode-ray-tube displays, and Flexowriters for direct output; analog-to-digital conversion equipment; data links with other computers; and miscellaneous special-purpose equipment. This paper will not be concerned with the details of these devices, but will limit itself to a discussion of the logical incorporation of them into the system.

In describing the TX-2 input-output system, reference will be made to certain design aspects of other parts of the TX-2 as set forth in the previous paper.

The Multiple-Sequence Program Technique

Of the various organizational schemes which permit the simultaneous operation of many devices, we have chosen the "multiple-sequence program technique" for incorporation in TX-2. A multiple-sequence computer is one that has several program (instruction) counters. If the program sequences associated with these program counters are arranged to time-share the hardware of the central computer, a machine can be obtained which will behave as if it were a number of logically separate computers. We call these logical computers *sequences* and therefore refer to TX-2 as a *multiple-sequence* computer. By associating each input-output device with such a sequence, we effectively obtain an input-output computer for each device.

Since the one physical computer in which these sequences operate is capable of performing only one instruction at a time, it is necessary to interleave the sequences if they are to operate simultaneously. This interleaving process can take place aperiodically to suit the needs of and under the control of, whatever individual input-output devices are operating. The number of sequences which can operate simultaneously, and the complexity of the individual sequences, is limited by the peak and average data-handling rate of the central computer hardware.

In a multiple-sequence computer, the main body of the computation can be carried out in any sequence, but if maximum efficiency of input-output operation is to be achieved, the bulk of arithmetic operations must be confined to a few special sequences, called *main* sequences, which have no associated input-output devices. The input-output sequences may then be kept short, and a large number of them can be executed at once.

Multiple-Sequence Operation in TX-2

In TX-2, one-half of the index-register memory has been made available for storing program counters. Thus, a total of 32 sequences may be operated in the machine. (Actually an additional sequence of special characteristics is obtained by using index register number 0 as a program counter. This special sequence will be discussed later.) Some of these sequences are associated with input-output devices. Others perform functions, such as interpreting arithmetic overflows, that are called into action by conditions arising within the central computer. Finally, there are the main sequences which are intended to carry out the bulk of the arithmetic computations performed by the machine.

A priority scheme is used to determine which sequence will control the computer at a given time. If more than one sequence requires attention at the same time, control of the machine will go to the sequence having the highest priority, and instructions addressed by its program counter will be executed.

Table I is a list of the sequences currently planned for inclusion in TX-2. They are listed in approximate order of priority with the highest at the top. Asterisks mark sequences which are not associated with any particular in-out device. A special sequence (number 0) has first priority and will be used to start any of the other sequences at arbitrary addresses. The next two sequences interpret alarms (under program control). These three sequences have the highest priorities, since they must be capable of interrupting the activities of other sequences. The input-output devices follow, with high-speed, free-running units carrying next highest priorities. The main sequences (we anticipate three) are at the bottom of the list. The priority of any sequence may be easily changed, but such changes are not under program control. Priorities are intended to remain fixed under normal operating conditions. The list totals about 25 sequences, leaving eight spaces for future expansion.

* The research in this document was supported jointly by the Army, Navy, and Air Force under contract with Mass. Inst. Tech.
† Lincoln Lab., M.I.T., Lexington, Mass.

TABLE I

TX-2 Sequence Assignments in the Order of Their Priority

*Start-Over (special index register number 0 sequence)
*In-out alarms
*Arithmetic alarms (overflows, etc.)
Magnetic tape units (several sequences)
High-speed printer
Analog-to-digital converter
Photoelectric paper tape readers (several sequences)
Light pen (photoelectric pick-up device)
Display (several sequences)
MTC (Memory Test Computer)
TX-0
Digital-to-analog converter
Paper tape punch
Flexowriters (several sequences)
*Main sequences (three)

* The sequences have no input-output device.

Switching between sequences is under the control of both the input-output devices (generalized to include alarms, etc.) and the programmed instructions within the sequence.

Once a sequence is selected and its instructions are controlling the computer, further switching is under control of the programmed instructions. Program control of sequence switching is maintained through two bits, called the *break* and *dismiss* bits, in each instruction. The *break* bit governs changes to higher-priority sequences. When the *break* bit permits a change, and some higher-priority sequence requests attention, a change will be made. The *dismiss* bit indicates that the sequence has completed its operation (for the moment, at least) and that lower-priority sequences may receive attention. The interpretation of the *break* and *dismiss* bits will be discussed in more detail.

The TX-2 Input-Output Element

The TX-2 input-output element is shown schematically in Fig. 1. It consists of a number of input-output devices, associated buffers, and a sequence selector. Each device has enough control circuitry to permit it to operate in some selected mode once it has been placed in that mode by signals from the central computer. Associated with each device is a buffer storage of appropriate size. This buffer may be large or small, to suit individual data-rate requirements, but the buffers used in TX-2 will generally be the smallest possible. For the most part, buffering for only one line of data from the device (*e.g.*, 6 bits for a paper-tape reader) will be provided. Each input-output device is associated with one stage of the sequence selector. The sequence selector provides the control information necessary for proper interleaving of the program sequences. When it is desired to add a new input-output device to the computer, the three packages, in-out unit, buffer, and sequence-selector stage, must be provided.

As shown in Fig. 1, data is transferred between the input-output element and the central computer by way of the exchange element. Fig. 1 indicates two-way paths between the E register and all in-out buffers. Actually,

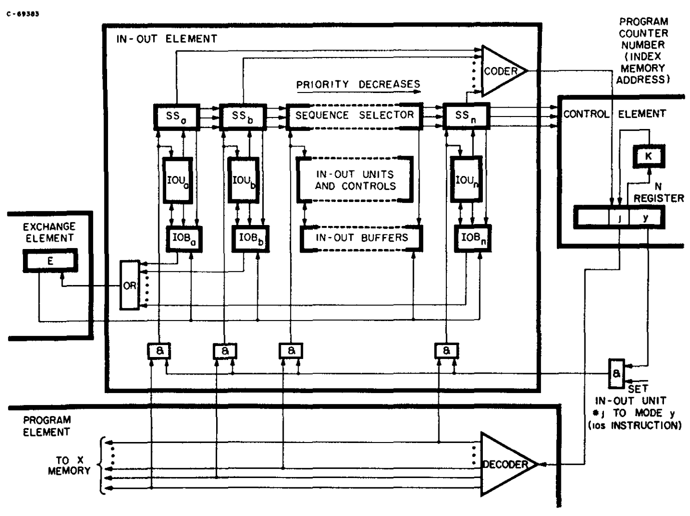


Fig. 1—Block diagram of TX-2 in-out element.

most devices are either readers or recorders, but not both, and therefore require one-way paths only. Only the necessary paths are provided; the drawing simply shows the most general case.

Signals from the sequence selector connect the appropriate buffer register to the E register to transfer data. When a sequence is selected (*i.e.*, its program counter is supplying instruction locations), the associated buffer is connected to the E register, and all other buffers are disconnected. A *read* instruction will effect a transfer of information between the buffer and the E register. A particular buffer is thus accessible only to *read* instructions in the sequence associated with the buffer's in-out unit.

Fig. 1 shows paths from the sequence selector to a coder which provides an output called the program-counter number. These paths are used in the process of changing sequences to be described in a later section.

Fig. 1 also shows paths for mode selection in the in-out element. The use of these paths is described in the next section under *ios*.

Input-Output Instructions

In addition to the break and dismiss bits on all instructions, the programmer has three computer instructions for operating the input-output system. There are two *read* instructions, *rdn* and *rds*, which transfer data between the in-out devices and the central computer memory. The third instruction, *ios*, selects the mode of operation of the in-out devices.

rdn and rds

Both of the *read* instructions obtain a word from memory. If the in-out device associated with the sequence in which the read instruction occurs is in a reading (input) mode, appropriate bits of the memory word are altered, and the modified word is replaced in memory. If the in-out device is in a recording (output) mode, appropriate bits of the memory word are fed to the selected in-out buffer, and the word is replaced in memory. Thus, the same *read* instruction suffices for both input and output operations. The distinction between *rdn* and *rds* lies in the assembling of full memory words from short buffer words. An *rdn* instruction will place the 6 bits from a tape reader in the right 6 bits of a 36-bit memory word. The remaining 30 bits will be left unchanged. An *rds* instruction for the same tape reader will place the 6 bits in a splayed pattern (every sixth bit across the memory word) and will shift the entire word one place to the left before replacing it in memory. Except for the shift, the other 30 bits remain unchanged. A sequence of 6 *rds* instructions, one for each of 6 tape lines and all referring to the same memory address, will suffice to assemble a full 36-bit word.

The distinction between *rdn* and *rds* could be obtained from mode information in the in-out device, but the inclusion of both instructions in the order code allows the programmer to interchange the two types freely to suit his needs. The *rdn* instruction makes use of the permutation aspect of the TX-2 configuration control and is, therefore, particularly convenient for dealing with alphanumeric Flexowriter characters. Configuration is not applicable to the *rds* instruction.

ios

The *ios* instruction serves to put a particular in-out device into a desired mode of operation. The *j* bits of the instruction word, normally the index register number, in this case specify the unit number of the in-out device. This number is the same as the program counter number for the associated sequence, although the correspondence is not necessary. The *y* bits of the instruction word specify the mode of operation in which the unit is to be placed. Two of the *y* bits are sent directly to the *j*th sequence selector stage and serve to control the sequence, regardless of the mode of its associated in-out device. These two bits allow *ios* instructions to arbitrarily dismiss or request attention for any sequence in the machine. By means of these instructions, one sequence can start or stop all others in the machine. A third *y* bit determines whether the mode of the in-out device is to change as a result of the instruction. If it is to change, the remaining 15 bits specify the new mode. An *ios* instruction occurring in any sequence can thus start or stop any sequence and/or change the mode of its in-out device.

A further property of the *ios* instruction is that it leaves in the *E* register a map of the state of the specified in-out control prior to any changes resulting from the instruction itself; *ios* instructions may, therefore, be used to sense the state of the in-out system without altering it in any way.

Sequence-Changing and Operation of the Sequence-Selector

At some point just before the completion of the instruction memory cycle in TX-2, the Control must decide whether the next instruction would be taken from the current sequence or from some new sequence. The information on which this decision must be based comes from the *break* and *dismiss* bits of the instruction word currently in use and from the sequence selector. Fig. 2 is a detailed drawing of one stage of the sequence selector. All stages, except that with the highest-priority, are identical. The lowest-priority stage returns the final three control signals to the control element.

Each stage of the sequence selector retains two pieces of information concerning its associated sequence. One flip-flop (ss j.1) remembers whether or not the sequence is selected (*i.e.*, whether or not it is receiving attention). The priority signal (labeled *no higher priority sequence requests attention*) passes from higher to lower priority stages until it encounters a stage which requests, but is not receiving attention. Such a stage is said to have priority at the moment, and its output to the program-counter-number coder prepares the number of the new program counter in anticipation of a sequence change.

The process of changing sequences involves storing the program counter for the old sequence and obtaining the counter for the new. Actually, to speed up the overall process, the new program counter is obtained first, so that it may be used while the old is being stored. Using the paths shown in Fig. 1, the new program counter number is placed in the *j* bits of the *N* register. The new program counter is then obtained from the *X* memory and interchanged with the old program counter contents which have been in the *P* register.[1] The *K* register, which has been holding the old program counter number since the last sequence change, is now interchanged with the *j* bits, and the old counter is stored at the proper location in the *X* memory. The state of the sequence selector is changed, to conform to the change of sequence, by sending a *select new sequence* command from Control. This command clears the ss j.2 flip-flop in the old-sequence stage and sets the ss j.2 flip-flop to a ONE in the new-sequence stage.[2]

Interpretation of the Break Bit

The programmer uses the *break* bit of an instruction word to indicate whether or not change to a higher priority sequence may occur at the completion of the instruction. The fact that a programmer permits a *break* does not mean that the sequence has completed its current task, but merely that no harm will be done if a change to some higher-priority sequence is made. *Breaks* should be permitted at every opportunity if a number of in-out devices are operating. The sort of situation in which a *break* cannot be permitted occurs when the *E* register is left containing information which the program requires at a later step. If a change occurred in this case, the contents of the *E* register would be destroyed and lost to the program.

[1] The *P* register is shown in Fig. 4 of Frankovich and Peterson, this issue, p. 148.

[2] The relative timing of the central computer actions during the change process is shown in Fig. 6(d) of Frankovich and Peterson, this issue, p. 150.

When a *break* is permitted by the current instruction, a sequence change will actually take place only if some higher-priority sequence requests attention. A signal from the sequence selector to the control element provides this information (Fig. 2). When a *break* type of sequence change is made, the ss j.1 flip-flop in the sequence selector remains unchanged, and the sequence which was abandoned in favor of one of a higher-priority continues to request attention.

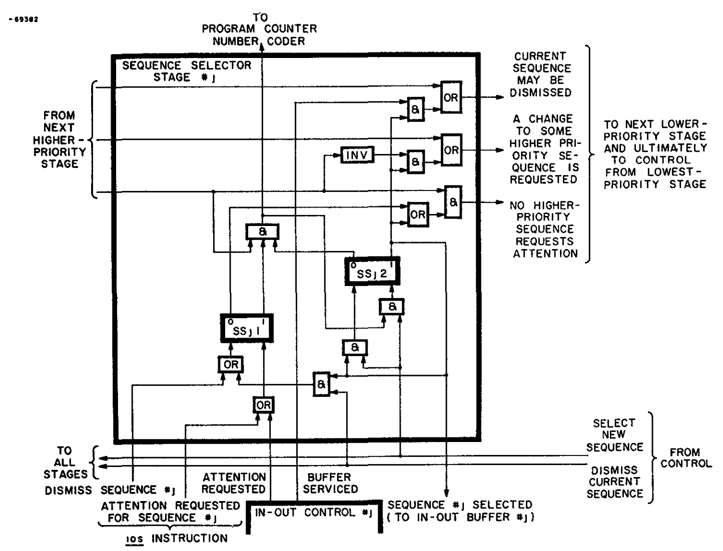

TO PROGRAM COUNTER NUMBER CODER
SEQUENCE SELECTOR STAGE #j
FROM NEXT HIGHER-PRIORITY STAGE
CURRENT SEQUENCE MAY BE DISMISSED
A CHANGE TO SOME HIGHER PRIORITY SEQUENCE IS REQUESTED
NO HIGHER-PRIORITY SEQUENCE REQUESTS ATTENTION
TO NEXT LOWER-PRIORITY STAGE AND ULTIMATELY TO CONTROL FROM LOWEST-PRIORITY STAGE
OR
a
INV
SSj 2
SSj 1
SELECT NEW SEQUENCE
DISMISS CURRENT SEQUENCE
FROM CONTROL
TO ALL STAGES
DISMISS SEQUENCE #j
ATTENTION REQUESTED
BUFFER SERVICED
ATTENTION REQUESTED FOR SEQUENCE #j
IN-OUT CONTROL #j
SEQUENCE #j SELECTED (TO IN-OUT BUFFER #j)
ios INSTRUCTION

Fig. 2—Block diagram of TX-2 sequence selector stage.

Interpretation of the Dismiss Bit

The *dismiss* bit is used by the programmer to indicate that the sequence presently in use has completed its task. To provide synchronization in the in-out system, *dismiss* bits must be programmed between attention requests from the in-out devices. In this case, the *dismiss* operation guarantees that the computer will wait for the next signal from the in-out device before proceeding with the associated program sequence.

The *dismiss* bit is also used to accomplish the halt function in TX-2. A multiple-sequence computer *halts* when all sequences have been dismissed and all in-out units turned off. The priority signal from the sequence selector to the control element provides the information as to whether or not any sequence in the machine requests attention. When none request attention, the control stops all activity in the machine as soon as a *dismiss* bit appears on an instruction in the sequence being used. Activity is resumed in the machine as soon as some in-out device or push button requests attention.

The sequence change which results from a *dismiss* bit is identical with that resulting from a *break* except that a *dismiss current sequence* command accompanies the *select new sequence* command from Control to the Sequence Selector (Fig. 2).

Starting a Multiple-Sequence Computer

In a single-sequence computer the starting process involves resetting the program counter to some arbitrary value and starting the control. In a multiple-sequence computer, the program counter for a particular sequence must be reset and the sequence started. In TX-2 a special sequence (number 0) has the highest priority and is used to facilitate starting. This sequence has the special feature that its program counter always starts at an initial memory location specified by a set of toggle switches. Attention for the sequence is requested by pushing a button on the console. By executing a short program stored in the toggle-switch registers of the *V* memory, this sequence can start (or stop) any other sequence in the machine. The starting process for an arbitrary sequence involves resetting its program counter by means of an *ldx* (load index register) instruction, and starting its sequence with an *ios* instruction.

The Arithmetic Element in Multiple-Sequence Operation

While efficient operation requires that the bulk of arithmetic operations be carried out in a main sequence, the arithmetic element in TX-2 is available to all sequences. Since once a change has been made to a higher-priority sequence, control cannot return to a lower-priority sequence until the higher-priority one has been dismissed, a simple rule allows the arithmetic element to be used in any sequence without confusion. If, whenever a higher-priority sequence requires the arithmetic element, it stores the contents of any registers it will need (*A*, *B*, *C*, *D*, or *F*) and reloads them before dismissing, all lower-priority sequences will find the registers as they left them. This storing and loading operation requires time and, therefore, lowers the total data-handling capacity, but the flexibility obtained may well be worth the loss in capacity.

The step-counter class of arithmetic element instructions is a special problem. These instructions can require many microseconds to complete, and while TX-2 is designed to allow in-out and program element instructions to take place while the arithmetic element is busy, the case can arise in which an arithmetic element instruction (load, store, etc.) appears before the *AE* is finished with a step-counter class instruction. The machine would normally wait in an inactive state until the operation is complete, but since there is a chance that some higher-priority sequence may request attention in the interim and have instructions which can be carried out, provision is made to keep trying changes to higher-priority sequences as they request attention. The machine thus waits in an inactive state only when no higher-priority sequences have instructions which can be performed. This provision allows the programmer to ignore the arithmetic element in considerations of peak- and average-peak rate calculations when he desires to operate a maximum number of in-out devices.

Conclusion

Multiple-sequence operation of input-output devices, as realized in TX-2, has a number of significant characteristics. Among them are:

1) A number of in-out devices may be operated concurrently with a minimum of buffering storage.

2) Machine time is used efficiently, since no time need be lost waiting for input-output devices to complete their operation. Other machine activity may proceed meanwhile.
3) Each input-output device may be treated separately for programming purposes. Efficiency of operation is obtained automatically when several separately programmed devices are operated simultaneously, although average- and peak-rate limitations must be considered.
4) Maximum flexibility in programming for input-output devices is obtained. The full power of the central machine may be used by each input-output sequence if desired. Routines for each device may be as long or as short as the particular situation requires.
5) The modular organization of the input-output equipment simplifies additions and modifications to the complement of in-out devices.
6) The organization of buffering storage allows the amount and kind of such storage to be tailored to the needs of the individual devices and the data-handling requirements to be met by the system.
7) The multiple-sequence program technique appears to be particularly well suited to the operation of a large number of relatively slow input-output devices of varying characteristics, as opposed to a smaller number of high-speed devices.

Memory Units in the Lincoln TX-2*

RICHARD L. BEST†

Memory Units in the Lincoln TX-2 Computer

THERE ARE 3 high-speed live memories in TX-2; all are random access and all use ferrite cores. The largest is the $6\frac{1}{2}$-μsec cycle time "S" memory with 65,536 37-digit words. The "T" memory is entirely transistor driven; it has a capacity of 4096 37-digit words and a $5\frac{1}{2}$-μsec cycle time. The smallest and fastest is the "X" memory with a capacity of 64 19-digit words; external word selection and 2 cores per bit make possible an access time of 0.8 μsec and a cycle time of 4 μsec.

"S" Memory (65,536 Words)

The "S" memory (Fig. 1) is a coincident-current magnetic core unit with a storage capacity of 65,536 37-bit words. The bits in the word are read out in parallel with a cycle time of 6.5 μsec and an access time of 2.8 μsec. (Cycle time is the time between successive strobe pulses and access time is the minimum delay between setting the address register and strobing.) The block diagram (Fig. 2, opposite) shows that two 256 position magnetic core switches are used to supply the READ and WRITE current pulses to the X and Y selection lines. The operating characteristics of these switches are such that the contents of the address register are no longer needed after the READ half of the cycle, and the interval between READ and WRITE may be extended several microseconds under computer control to permit the other operations to occur. Two coordinates are used to select a register during READ and three coordinates are used for WRITE. In each case, 2:1 current selection ratio is used. The S memory with 604 tubes and 1406 transistors, is a 37-digit version of the 19-digit TX-0 memory that has been described in the literature.[1] The basic operation of this type memory has also been described and will not be repeated here.[2]

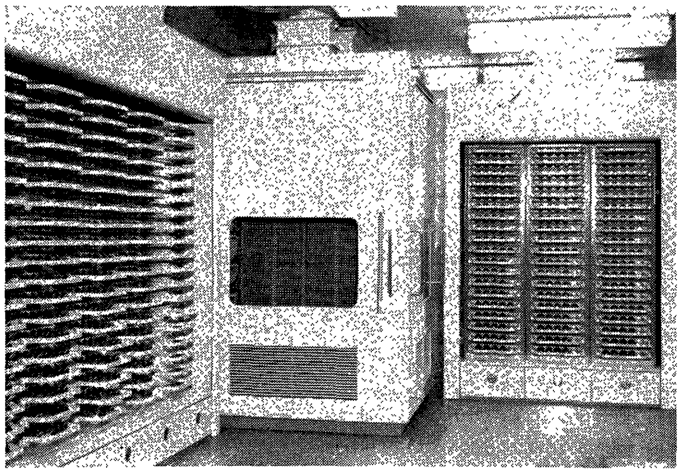


Fig. 1—"S" memory, 65,536 words, 37 digits.

* This work was supported jointly by the Army, Navy, and Air Force under contract with the Mass. Inst. Tech.
† M.I.T. Lincoln Lab., Lexington, Mass.

[1] J. L. Mitchell, "Part I, the TX-0 memory," *Proc. Eastern Joint Computer Conference;* December, 1956.
[2] J. W. Forrester, "Digital information storage in three dimensions using magnetic cores," *J. Appl. Phys.*, vol. 22, pp. 44–48, January, 1951.

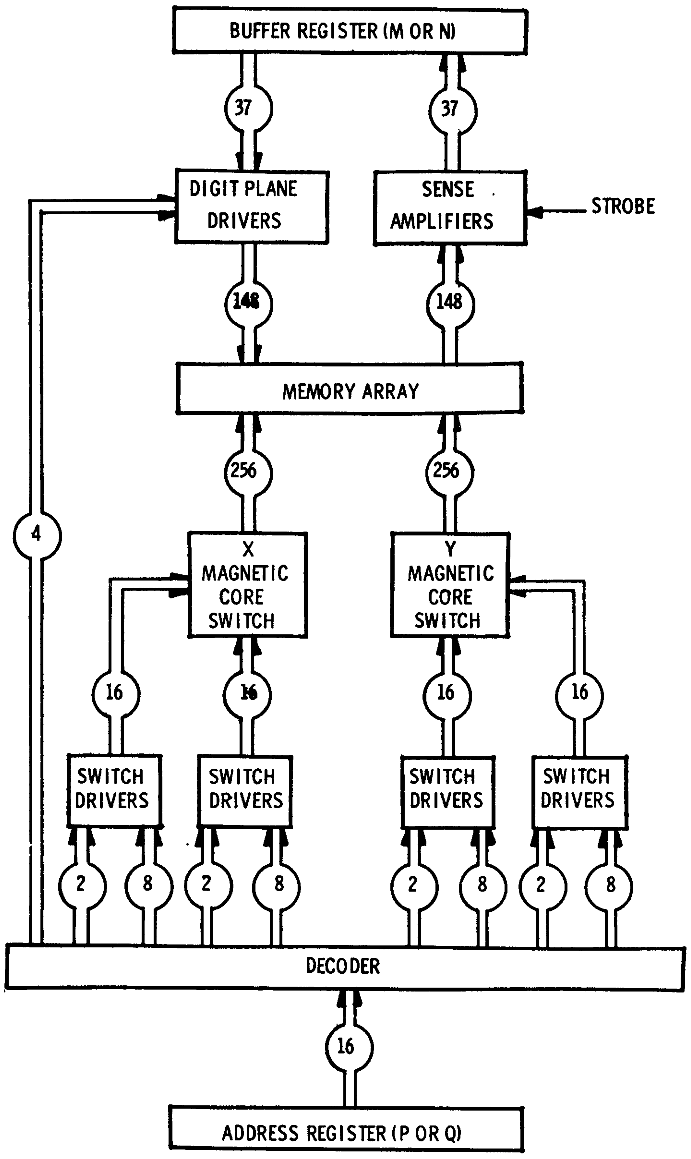


Fig. 2—Block diagram, "S" memory.

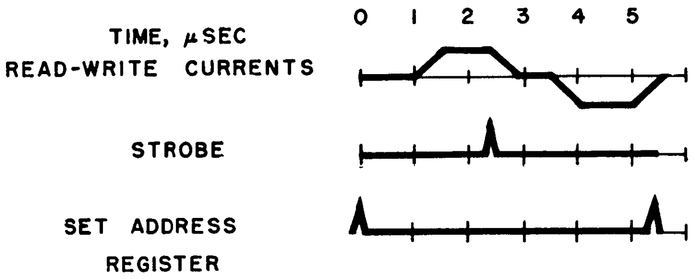


Fig. 3—Timing, "T" memory.

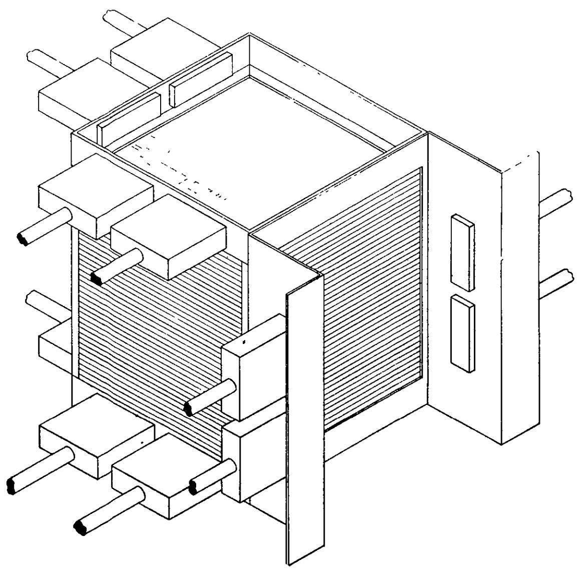


Fig. 4—"T" memory, 4096 words, 37 digits, $4\frac{5}{8}\times4\frac{5}{8}\times5$ inches inside the brackets.

"T" Memory (4096 Words)

The 4096 37-bit word memory is also a coincident-current magnetic core unit. The bits of the word are read out in parallel with a cycle time of 5.5 μsec and an access time of 2.4 μsec. A timing diagram is shown in Fig. 3. The computer program can extend by any amount the interval between READ and WRITE. Again, a 2:1 current selection ratio is used with two coordinates used to select for READ and three coordinates for WRITE. A total of 1460 transistors and 64 diodes are used (not counting the address and buffer registers and control).

Mechanical Features

The 64×64×38 (one spare plane) pluggable array (Fig. 4) is contained within a 5-inch cube. The cores used at 47 mils OD, 27 mils ID, and 12 mils thick. The material is similar to General Ceramics' S-1, and is also used in the S memory.

Selection Circuits

The logic and circuitry of the memory address register decoder is shown in Figs. 5 and 6 (next page). The input to the emitter follower AND gates is a dc level of zero or −3 volts. Silicon diodes add a bias shift without the loss that would be associated with a simple voltage divider. The +1.2 volt supply for the inverter AND gates is a single divider for the whole memory—the load on the divider is constant.

Each inverter AND gate feeds a selection line driver (Fig. 7). Q4 passes the full selection line current (+ and −250 ma) and is selected to have a minimum β of 10 for current of either polarity. The transient back voltage of the selection line for this current is 12 volts. The series connected emitter followers (Q1 and Q2) supply the large amount of current needed to cut off Q3 quickly during selection.

The load for Q2 is such that a large surge of current is delivered to Q3 to turn it on quickly when this line is deselected.

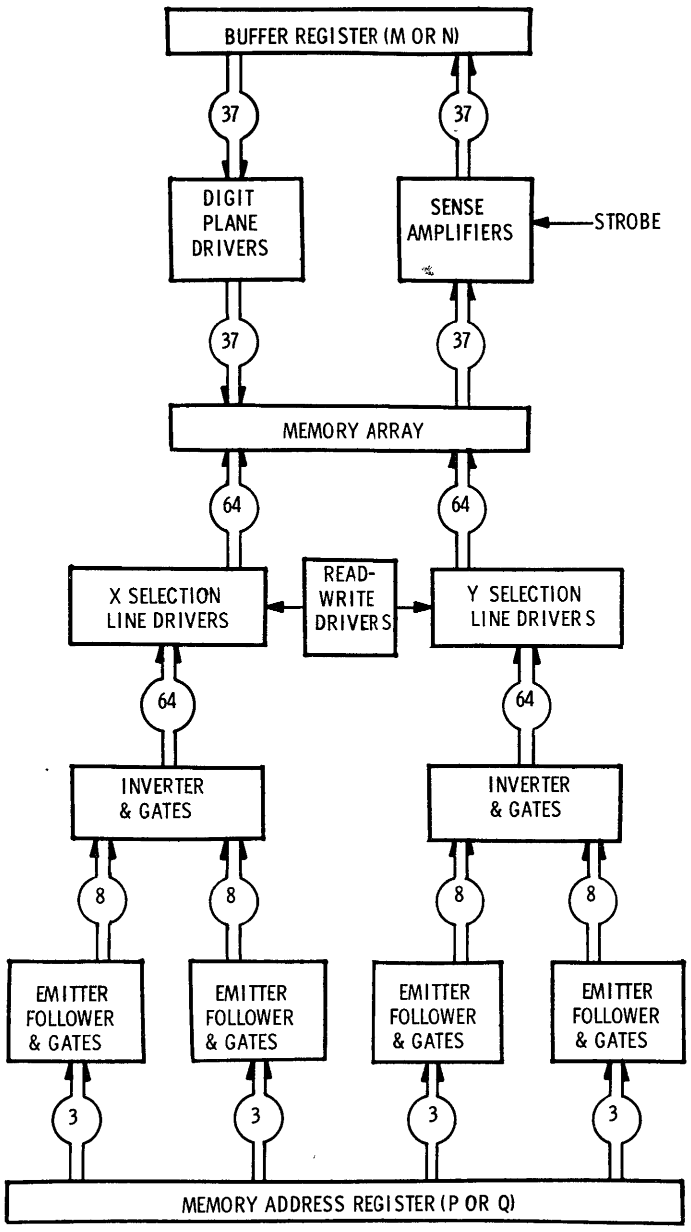


Fig. 5—Block diagram, "T" memory.

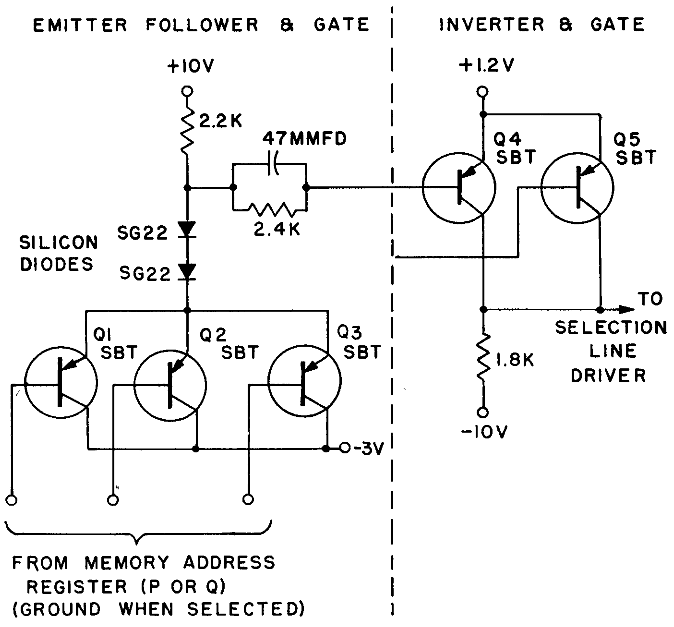


Fig. 6—One channel, emitter follower and inverter AND gates, "T" memory.

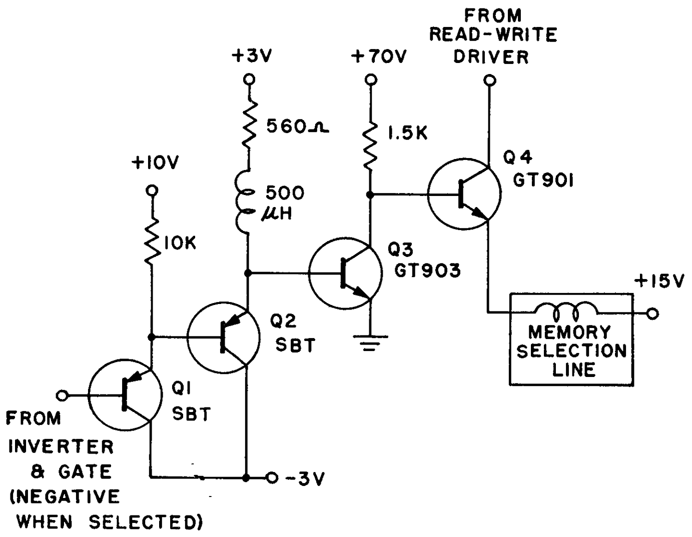


Fig. 7—One channel, selection line driver, "T" memory.

Fig. 8 (opposite) shows the circuit geometry planned for the read-write driver. Currents for READ and WRITE operations are supplied by the 150-volt supplies through R2 and R1 respectively. The currents are switched into the proper selection line drivers by cutting off Q1 or Q2.

Digit Circuits

The input of the digit plane driver shown in Fig. 9 is a standard logic level of 0 or −3 volts. Q2 acts as a switch which connects a voltage source across the digit winding and the parallel RC combination. It can be turned on and off very quickly by virtue of the large overdrive of current into its base which is supplied by the combination of Q1 and its collector load. The adjustable resistor is used to set the correct dc inhibit current which is measured across the 2-ohm resistor, and the 0.001-μf capacitor is used to speed the current rise time in the digit winding.

In the sense amplifier shown in Fig. 10 two 160-ohm resistors terminate the sense winding and tie it down to ground. Constant dc emitter currents are supplied to Q1 and Q2 by the 13k resistors. A stable dc collector-to-base voltage results from the voltage divider comprising the 1.3k resistors which form a virtual center tap on the sense winding operating in conjunction with the 3.3k resistor. Thus, with the dc emitter current and the base-to-collector voltage stabilized, the operating point of Q1 and Q2 is stabilized. Two 60 μf electrolytic capaci-

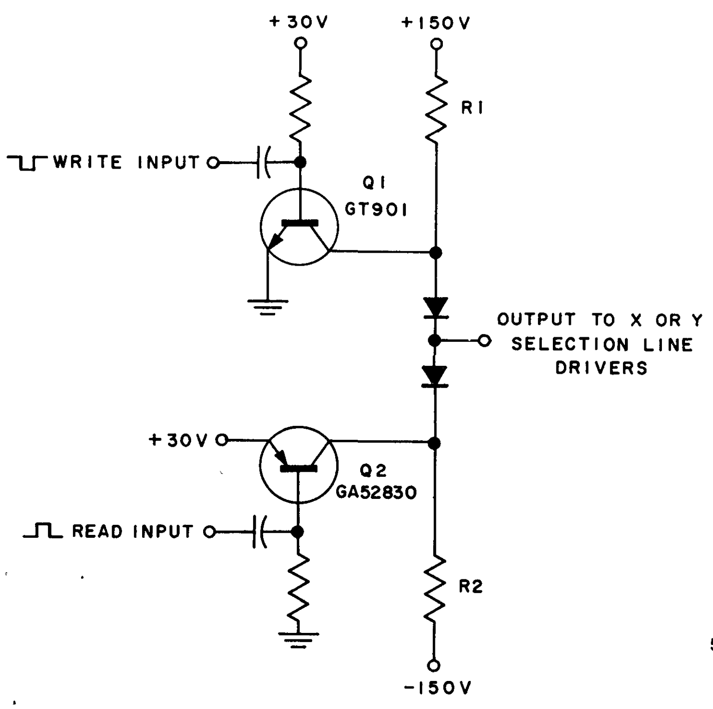


Fig. 8—Read-write driver (2 needed), "T" memory.

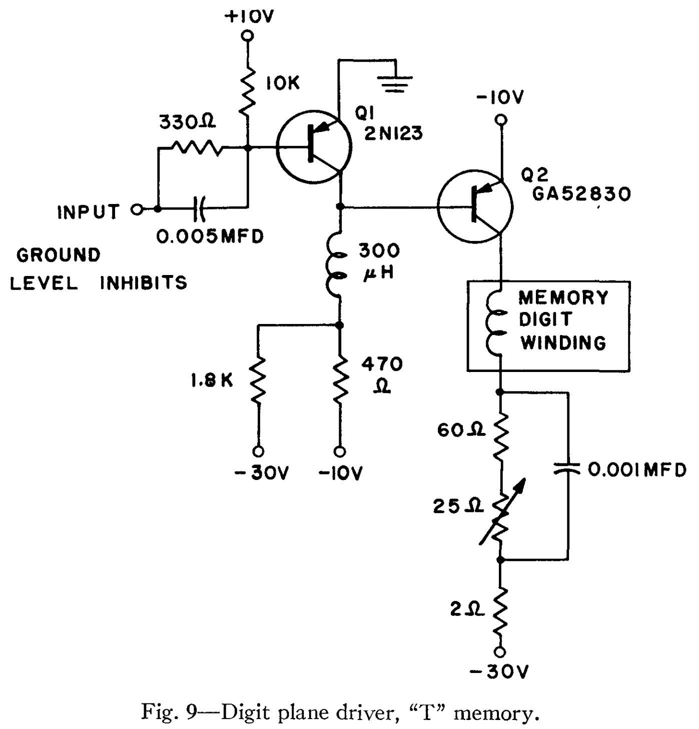


Fig. 9—Digit plane driver, "T" memory.

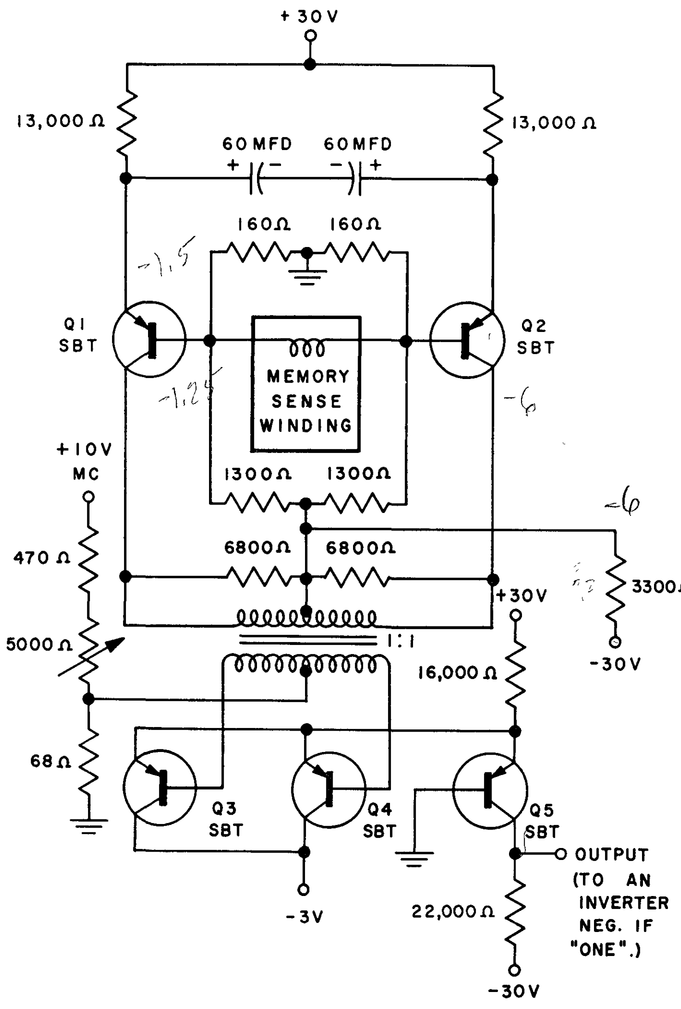


Fig. 10—Sense amplifier, "T" memory.

tors in series tie the emitters of Q1 and Q2 together for signal gain. The 6.8k resistors damp the transformer windings.

There is no gain for a common mode input, and a gain of about 22 for a difference-signal input (output measured across half the transformer secondary). The current through the 16k resistor normally flows through Q5 but can be interrupted by an input signal large enough to overcome the bias on the 68-ohm resistor. The 5k variable resistance is adjusted so that a 50-mv input signal will be just enough for this purpose. The normal ONE input signal is 100 mv. All of the +10 volt marginal check lines are tied together so that the sense-amplifier clip levels may be remotely checked to determine the memory margins.

"X" Memory (64 Words)

There are three modes of operation of the "X" memory:

1) READ-WRITE,
2) READ, and
3) CLEAR-WRITE.

The external-word-selection magnetic-core unit using 2 cores per bit has a storage capacity of 64 19-bit words. The bits of the word are read out in parallel with a cycle time of 4 μsec and an access time of 0.6 μsec. In this memory, cycle time is the time between successive strobe pulses with a repetitive READ-WRITE cycle; access time is again the minimum delay between setting the

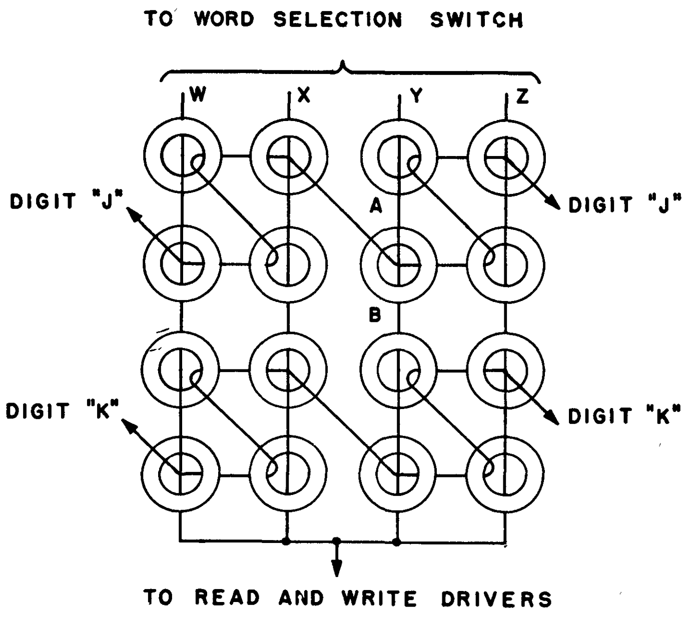


Fig. 11—Winding configuration, "X" memory.

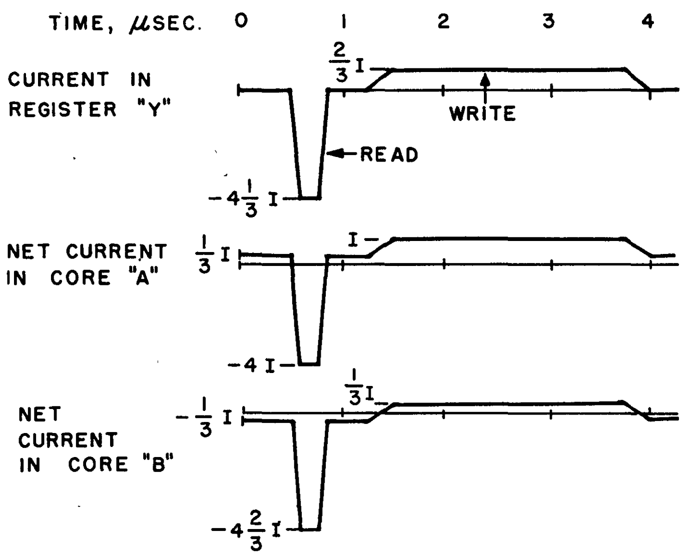


Fig. 12—Timing diagram, "X" memory.

address register and strobing. A total of 434 transistors, 8 diodes, and 1 vacuum tube are used excluding the address and buffer registers and control.

Operating Principle

The winding configuration of the single plane unit is shown in Fig. 11. A word is selected externally by connecting the upper end of a word line (pt. *Y*, for instance) to a fixed point. The READ driver then puts out a current pulse $4\frac{1}{3}$ times that required to switch a core on a 2:1 basis (Fig. 12). Only one of the two cores (per bit) is switched to the cleared state by this pulse because any previous WRITE operation would have left one core set, and one cleared. The switched core generates a pulse in its digit line. This line passes through one of the cores in the same direction as the word line and through the other core in a direction opposite to the word line. Thus, the polarity of the pulse on the digit line during READ, indicates whether a ONE or a ZERO is being read out.

Current always flows in the digit winding. The polarity is controlled by the flip-flop associated with that digit, and the amplitude is $\frac{1}{3}$ of the required switch current in a 2:1 system. The digit current is swamped out by the large read current and therefore has no effect during READ. During WRITE, a current of $\frac{2}{3}$ is sent down the selected word line. The digit current adds to the write current in one core and subtracts from it in the other, so that one core has a current of unity and the other a current of $\frac{1}{3}$. Thus, the current ratio used during WRITE is 3:1 with a disturb current of no more than $\frac{1}{3}$. Fig. 12 shows the timing and current relationships in cores *A* and *B* of Fig. 11.

The cores used for the "X" memory are 47 mils OD, 27 mils ID, and 12 mils thick. The core material is similar to General Ceramics' type S-3 which differs from S-1 in that the coercive force required for switching is lower. and the switching time is longer. We needed the low coercive force so that we could drive the cores with transistors.

Access time remained short because, even with transistors, we could overdrive the cores during READ. Each winding makes 4 turns on each core through which it passes. Fig. 13 (opposite) shows the complete memory plane ($4\frac{1}{4}\times 6\frac{1}{4}$ inches) and Fig. 14 shows a portion of it enlarged. The cores are mounted on a lucite plane; the wires pass through openings made by the intersection of milled slots on one side of the plate with similar slots on the other side milled at right angles to the first. With each winding making 4 turns per core, the digit current is 8 ma, the write-driver output current is 18 ma, and the read driver current is 117 ma.

The block diagram is shown in Fig. 15. The particular method of word selection used is determined partially by the computer's use of the outputs of the *j*-bits decoder. Two write drivers are used and the output of the first level selection determines which one is used.

Selection Circuits

One channel of the selection circuit is shown in Fig. 16. The *j*-bits decoder uses 5-way emitter follower AND gates which drive parallel inverters (Q6 and Q7). The collector load of these transistors is such as to provide an overdrive of base current into Q8 or Q9 during both selection and deselection. When neither read nor write driver is active, the word lines are free to float between 0 and −10 volts. Only one of the first-level selection transistors (Q10 or Q11) will be saturated, so base current flows only into either Q8 or Q9. The read driver generates a negative pulse so that the large read current (117 ma) flows in the normal direction in the 2N123's.

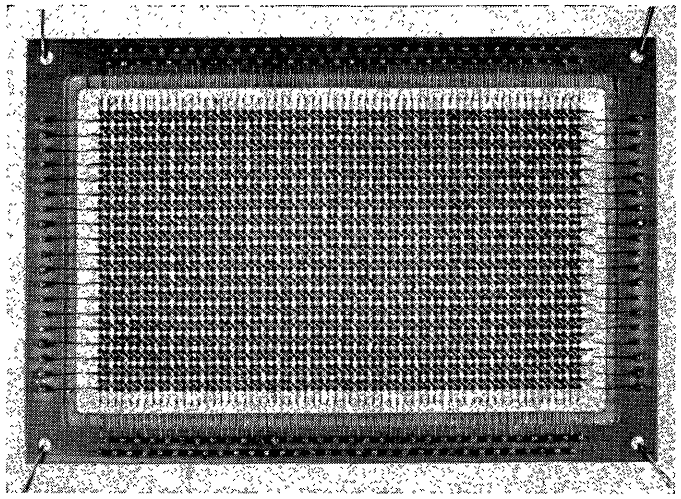


Fig. 13—"X" memory plane, complete ($4\frac{1}{4}\times6\frac{1}{4}$ inches over-all).

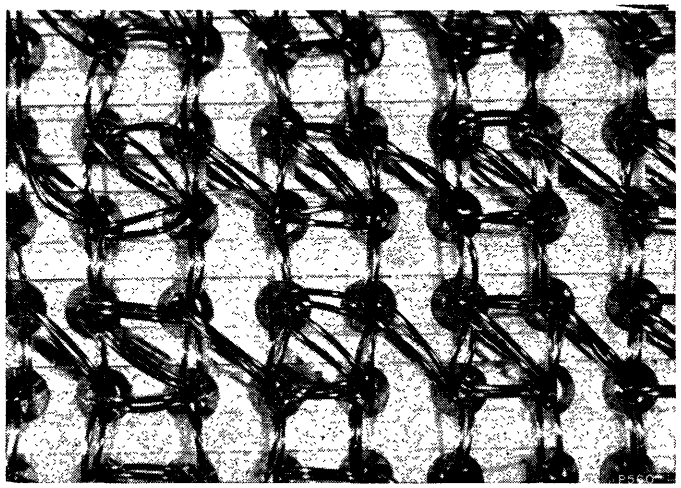


Fig. 14—"X" memory plane enlarged.

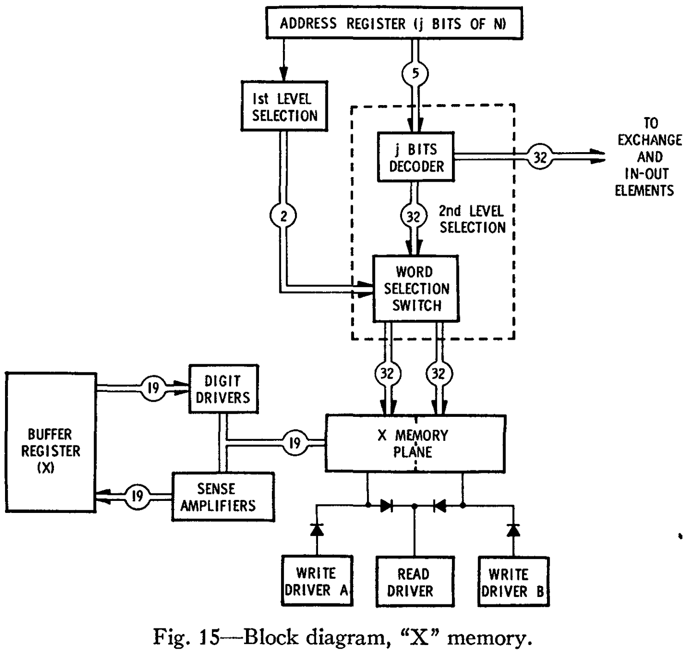


Fig. 15—Block diagram, "X" memory.

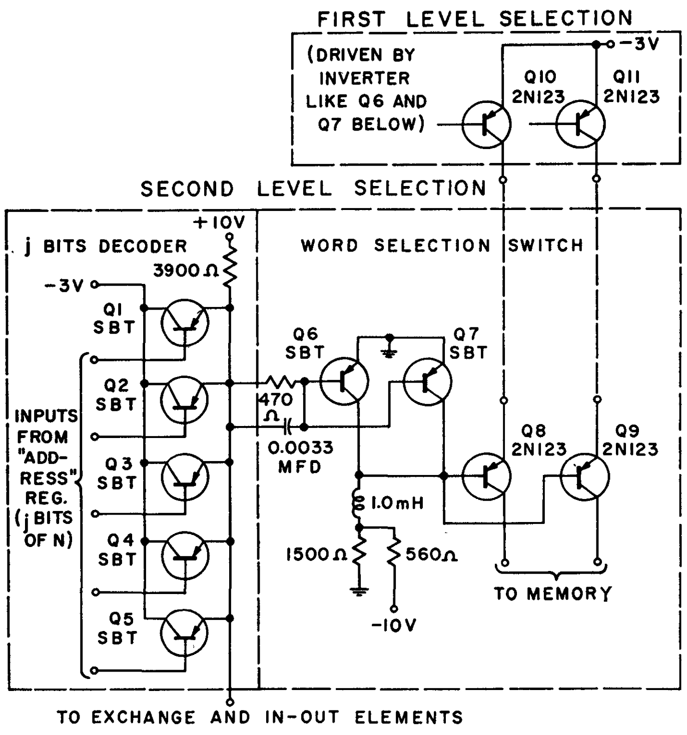


Fig. 16—Register selection circuit, "X" memory.

The write current flows in the reverse direction, but it is only 18 ma, and does not require a very high reverse β. It would have been more economical of transistors to use a decoder such as that in the "T" memory, but access time is at a premium here, so that the faster circuit was used.

Read-Write Drivers

The read driver shown in Fig. 17 (next page) consists of three SBT transistors in series (because of the voltage needed) driving a 6197 to saturation. The back voltage presented by the cores to this driver is constant because, as mentioned before, it always switches one of the two cores in each pair. The 5:1 transformer holds the tube load to a low value.

The write driver (Fig. 18) is very simple—the current in the 1640-ohm resistor is switched into the memory load during WRITE by saturating Q2 which cuts off Q1. Since the selection circuits are returned to -3 volts, the output terminal of this circuit is always below ground.

Digit Circuits

The digit driver (Fig. 19) is connected directly to the corresponding flip-flop in the buffer X register. One of the two transistors is always saturated so that current always flows in the digit winding and in a direction determined by the flip-flop. The terminals of the digit winding are connected to input stage (Q1 and Q2) of the sense amplifier shown in Fig. 20 which responds to the voltage difference between the inputs. The open circuit

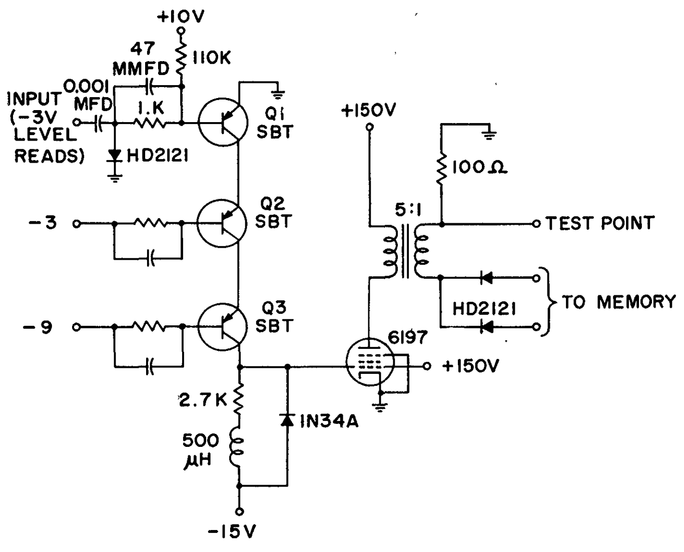


Fig. 17—Read driver, "X" memory.

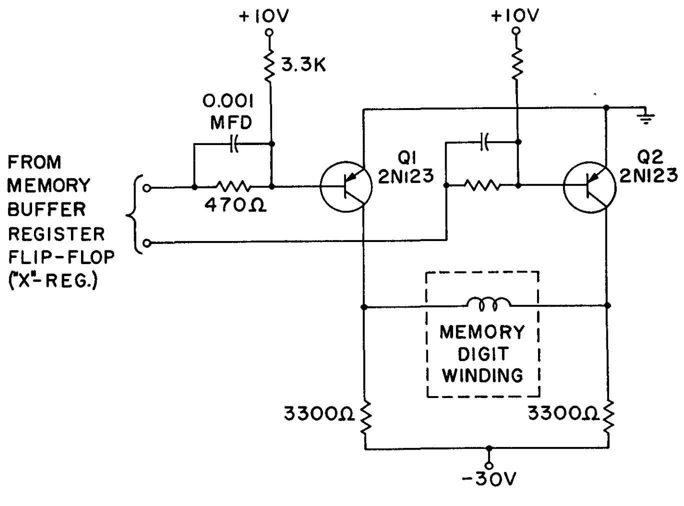


Fig. 19—Digit driver, "X" memory.

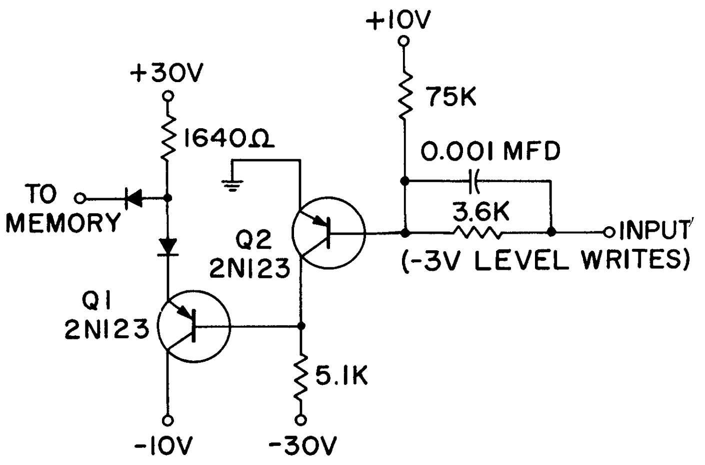


Fig. 18—Write driver, "X" memory.

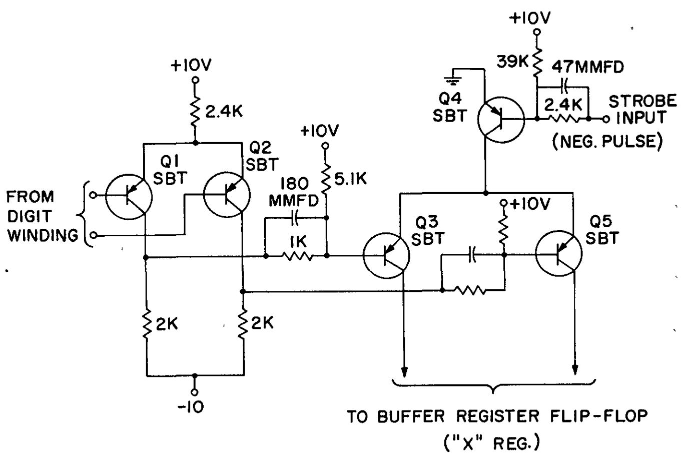


Fig. 20—Sense amplifier, "X" memory.

READ signal on the digit winding is a $\frac{1}{4}$-μsec pulse $\pm\frac{1}{2}$ volt in amplitude. The sense amplifier loads the winding to reduce the pulse to about half this amplitude. A saturation signal is fed to the gates Q3 and Q5 so that the strobe pulse forces the flip-flop to the correct position. If the signal on the free end of the digit winding is positive the flip-flop is left in the same state; if it is negative the flip-flop is complemented.

Modes of Operation

There are three modes of operation of the X memory:

1) READ-WRITE,
2) READ, and
3) CLEAR-WRITE.

READ-WRITE has been described above. The READ operation, used when the contents of two registers are needed quickly, performs the necessary function of clearing both cores in each bit before writing. When the computer returns to WRITE in registers that have had a READ cycle only, the CLEAR-WRITE cycle is used. CLEAR-WRITE is the same as READ-WRITE except that the strobe pulse is eliminated. Actually, a WRITE cycle alone would be sufficient but the CLEAR-WRITE cycle was added as an aid to program trouble shooting, since if a WRITE operation should follow a previous WRITE operation on the same register, some bits would have both cores set. A subsequent READ would clear both cores, their outputs would subtract in the digit winding, and the response of the sense amplifier would be unpredictable.

Acknowledgment

The results reported above were due to the efforts of many people associated with core-memory development at Lincoln Laboratory.

Major contributions to the system and circuit design were made by S. Bradspies, G. A. Davidson, D. H. Ellis, and J. L. Mitchell. E. A. Guditz was responsible for most of the ideas incorporated in the mechanical design and packaging.

Discussion

D. J. Theobold (U.S.N.E.L.): What type of core material was used?

Mr. Best: The cores for all our memories are made at M.I.T. so we do not have direct counterparts. Two larger memories use the materials which are quite similar to General Ceramics' S-1. It switches in one microsecond. The core material used in the index memory is a very low-gravity, coarse material that is not square enough to be used in the two-to-one selection. It switches with a driving current of 110 milliamp-turns.

E. E. Jungclas, Jr. (Hughes Aircraft): What are the operational temperature limits?

Mr. Best: The relatively high-speed core is used in the two larger core memories, with a high enough Curie temperature for most land base applications. The index memory that is used has a core that has quite a bit of zinc in it, in order to get the cores to force down. The Curie temperature of that is relatively low.

Jan Rajchman (RCA Laboratories): What is the inside diameter of the 0.047-inch cores?

Mr. Best: The inside diameter is 27 mills.

David Zeheb (General Electric): Would you amplify on the manner in which you use two cores per bit in the index memory?

Mr. Best: The two cores used in this particular bit are *A* and *B* (Fig. 10). The reason for using two cores is so that during "read" one can overdrive the cores very heavily, and switch them quickly, and therefore get short access time. You can only use two cores per bit when you have external selection, that is, some external active element for each word. The read current only goes through these cores, and it does not disturb any portion in the whole range. The main reason for going into the two cores per bit is to get a short read time.

When you have two cores there are actually four possible states of those cores; you can have both clear, or both set, right after you have read you have both cores cleared. But you never have both cores set, at least never on purpose. The primary reason for going into two cores was to get a fast read time (indicating on slide). A current of $\frac{2}{3}$ is set in the word line and a current of $\frac{1}{3}$ in the digit winding. These two windings are wound so that they add in one core and subtract in another. So, depending upon the polarity of the current in the digit winding, only one of the two cores would be set.

R. L. Compton (Librascope, Inc.): Do you mean to imply that the magnetic core memory system was operative to temperatures of the same order as the Curie temperature of the cores?

Mr. Best: No. The room is air-conditioned.

Transistor Circuitry in the Lincoln TX-2*

KENNETH H. OLSEN†

Circuit Configurations

Only two basic circuits are needed to perform most of the logical operations in the TX-2 computer; a saturated transistor inverter and a saturated emitter follower. To the logical designer who works with them, these circuits can be considered as simple switches which are either open or closed.

The schematic diagram of an emitter follower and the symbol used by the logical designers is shown in Fig. 1.

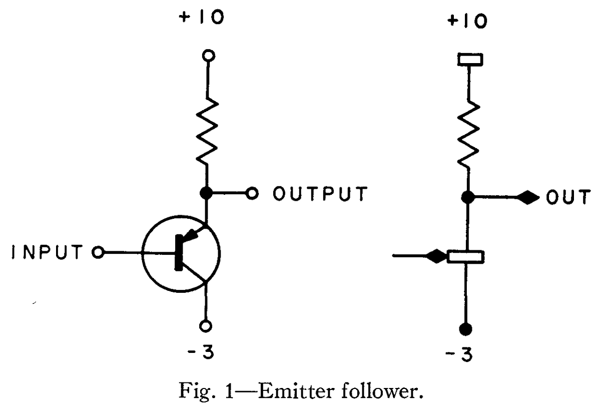


Fig. 1—Emitter follower.

With a negative input, the output is "shorted" to the -3-volt supply as through a switch. When several of these emitter followers are combined in parallel, as in Fig. 2, any one of them will clamp the output to -3 v.

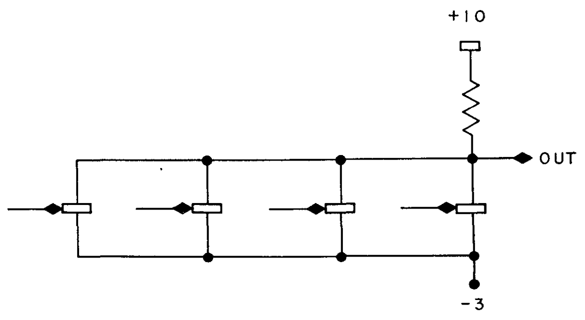


Fig. 2—Parallel emitter follower.

We have then an OR circuit for negative signals and an AND circuit for positive signals. The transistor inverter is shown in Fig. 3 (next page) with its logic symbol. Basic AND, OR circuits result from the connection of these simple switches in series or parallel (Figs. 4 and 5). More complex networks like the TX-2 carry circuit use these elements arranged in series-parallel (Fig. 6).

In Fig. 3 the resistor R_1 is chosen so that under the worst combinations of stated component and power

* This work was supported jointly by the U. S. Army, Navy, and Air Force under contract with Mass. Inst. Tech.
† Lincoln Lab., M.I.T., Lexington, Mass.

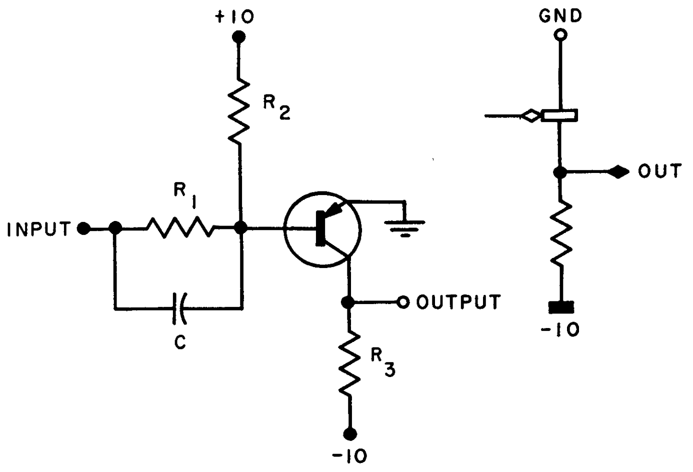


Fig. 3—Inverter.

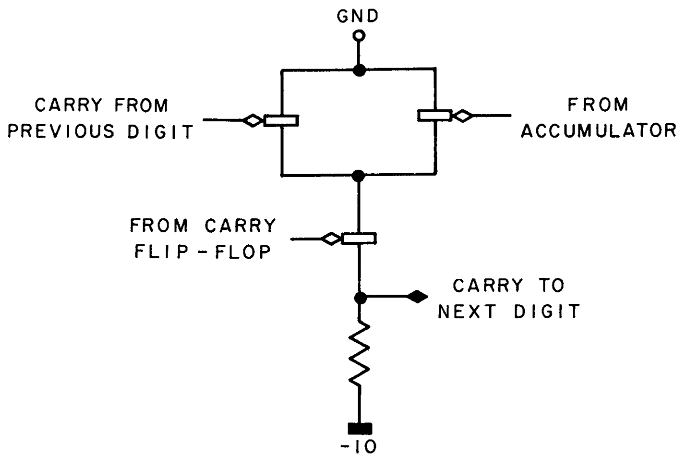


Fig. 6—TX-2 carry circuits.

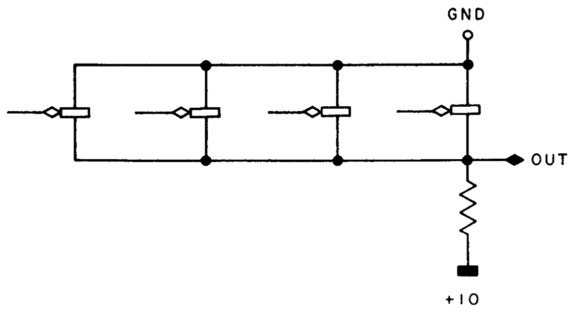


Fig. 4—Parallel inverters.

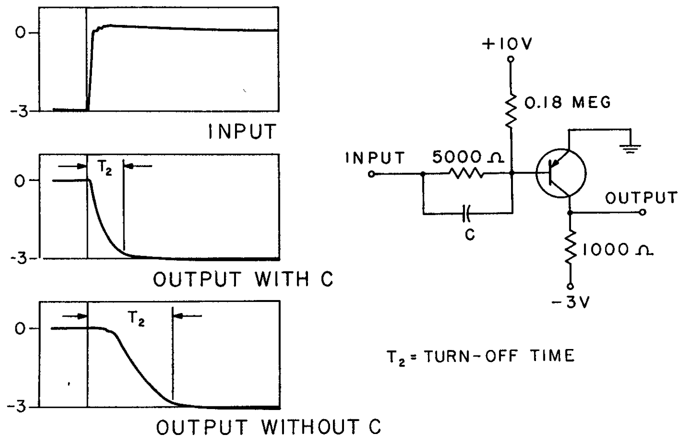


Fig. 7—Turn-off time.

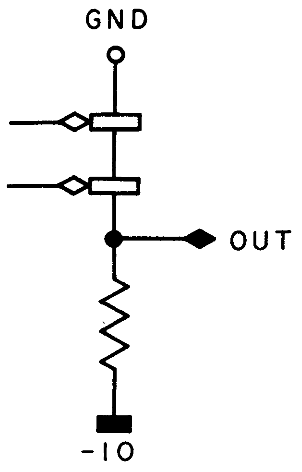


Fig. 5—Series inverters.

supply variations, the drop across the transistor will be less than 200 millivolts during the "on-condition." R_2 biases the transistor base positive during the off condition to provide greater tolerance to noise, I_{co}, and signal variations. Capacitance C was selected to remove all of the minority carriers from the base when the transistor is being turned off. The effect of C on a test circuit driven by a fast step is shown in Fig. 7. Note that the delay due to hole storage is only a few millimicroseconds.

We run the circuits under saturated conditions to achieve stability and a wide tolerance to parameters without the need for clamp diodes. Unlike vacuum tubes which always need an appreciable voltage across them for operation, a transistor requires practically no voltage across it. In spite of the delay in turning off saturated transistors, these circuits are faster than most vacuum tube circuits. Faster circuit speed is not due to the fact that the transistors are faster than vacuum tubes, but because they operate at much lower voltage levels. A vacuum tube takes a signal of several volts to turn it from fully "on" to fully "off;" a transistor takes less than one volt.

Flip-Flop

On the basis of previous experience, we decided that the advantages of having one standard flip-flop were worth some complication in TX-2 circuitry. The circuit diagram of the flip-flop package in Fig. 8 is basically an Eccles-Jordan trigger circuit with a three-transistor amplifier on each output. The input amplifiers isolate the pulse input circuits and give high input impedance. The amplifiers give enough delay to allow the flip-flop to be set at the same time that it is being sensed. Fig. 9 shows the waveforms of this flip-flop package when complemented at a 10-megapulse rate. The rise and fall

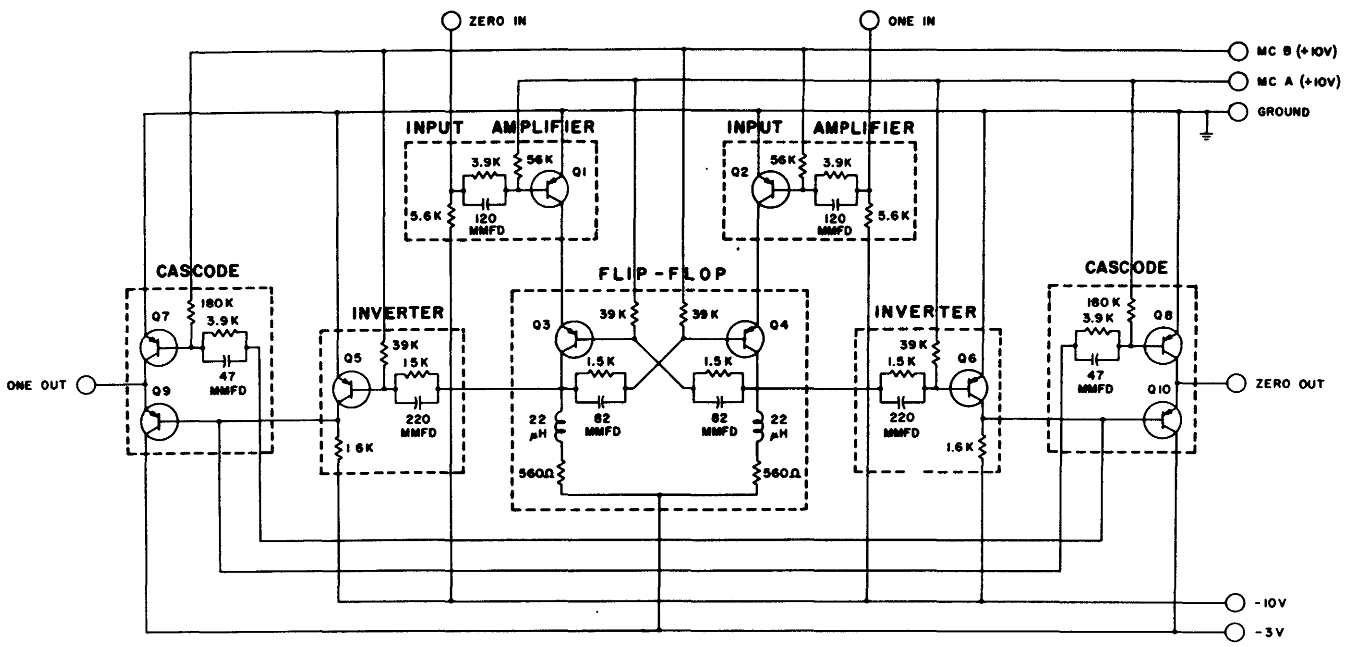


Fig. 8—TX-2 flip-flop.

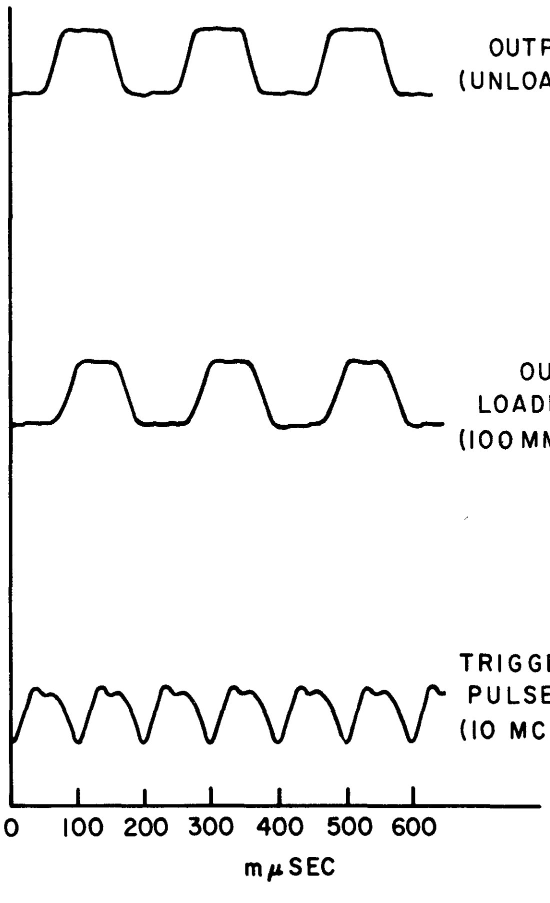


Fig. 9—Flip-flop waveforms.

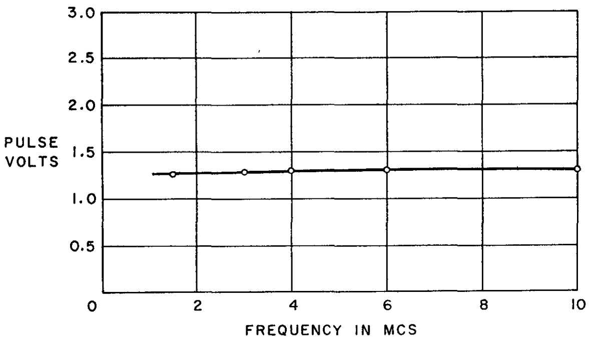


Fig. 10—Trigger sensitivity.

times, about 25 millimicroseconds, are faster than one normally sees in a single inverter, or an emitter follower because on each output there is an inverter that pulls to ground and an emitter follower that pulls to −3 v.

Fig. 10 is a plot of the pulse amplitude necessary to complement the flip-flop at various frequencies. Note the independence of trigger sensitivity to pulse repetition rate. This circuit will operate at a 10-megapulse rate, twice the maximum rate at which it will be used in TX-2.

The TX-2 circuits reproduced most often were designed with a minimum number of components to achieve economies in manufacture and maintenance. The design of less frequently reproduced circuits made liberal use of components—even redundancy to achieve long life and broad tolerance to component variations. The goal was system simplicity and high performance with a lower total number of components than might otherwise be possible. For example, the number of flip-flops in the TX-2 is small compared to the gates which transfer information from one group of flip-flops to another; so the flip-flops were allowed to be relatively complicated but the TX-2 transfer gates were made very simple. A transfer gate is only a single inverter. The emitter is connected to the output of the flip-flop be-

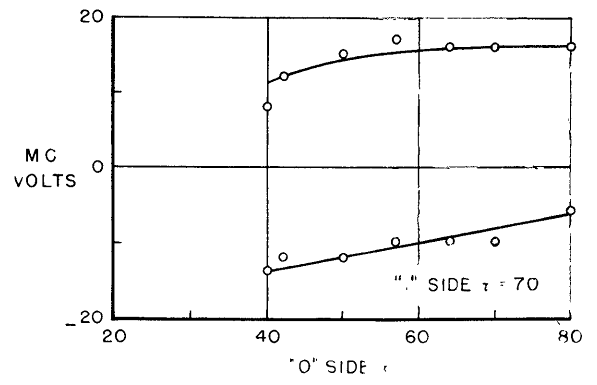


Fig. 11—Tau margins.

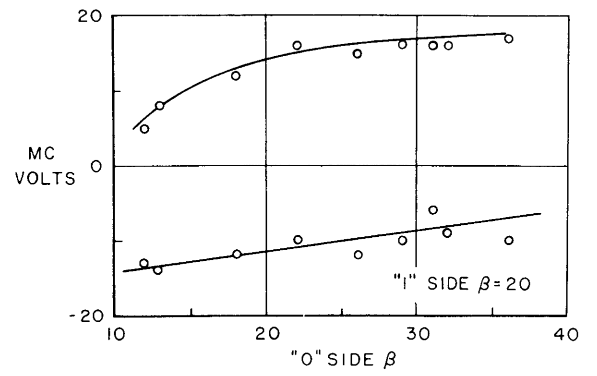


Fig. 12—Beta margins.

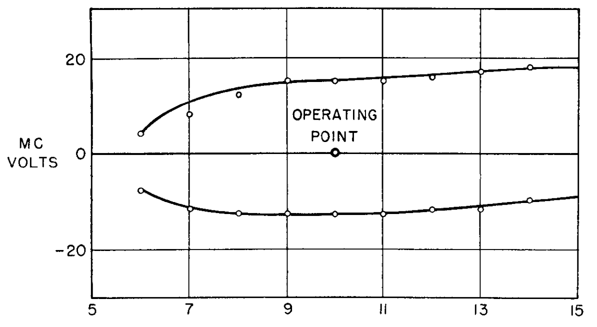


Fig. 13—–10-volt supply margins.

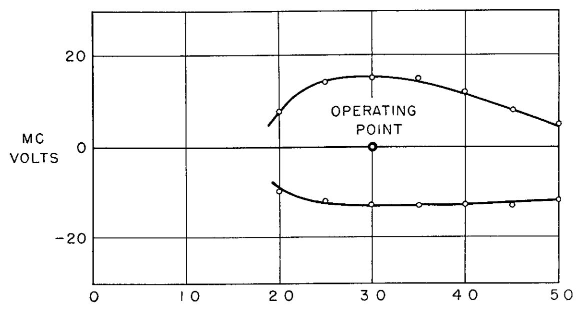


Fig. 14—–3-volt supply margins.

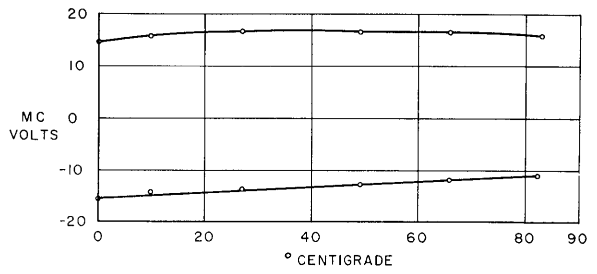


Fig. 15—Temperature margins.

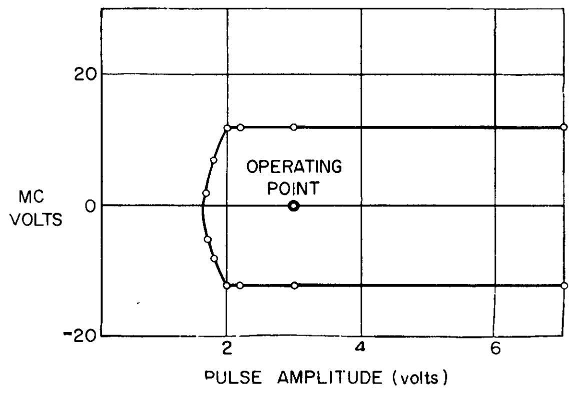


Fig. 16—Pulse margins.

ing read and the collector is connected to the input of the flip-flop being set. The output impedance of the flip-flop is so low that, when the output is at the ground level, a pulse on the base of the transfer gate shorts the input of the other flip-flop to ground and sets its condition.

Marginal Checking

We planned, of course, to incorporate marginal checking in the design of these circuits so that, under a program of regularly scheduled maintenance, deteriorating components could be located before they caused failure in the system. We also found it practical to use the technique during the design of the circuits to locate the design center of the various parameters and to indicate the tolerance of circuit performance to these parameters. A further application of marginal checking has been found in other systems during shakedown and initial operation to pin point noise and other system faults not serious enough to cause failure and therefore very difficult to isolate by other means.

The operating condition of the inverters is indicated by varying the +10-v bias. In the flip-flop schematic in Fig. 8, the inverters were divided into two groups for marginal checking, and the two leads labeled MCA and MCB were varied one at a time for most critical checking of the circuit. The following curves show the locus of failure points for various parameters as a function of the marginal checking voltage. Fig. 11 shows the tolerance to tau, a measure of hole storage and Fig. 12 shows the tolerance to beta, the current gain. Operating margins for supply voltages, temperature, and pulse amplitude are shown in Figs. 13 through 16.

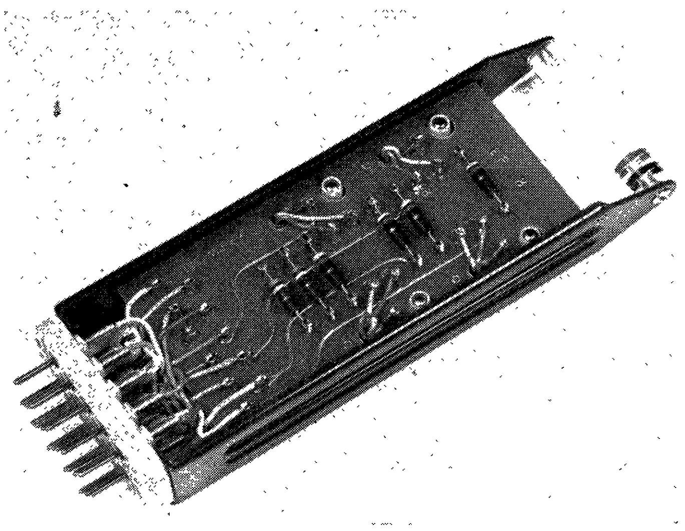


Fig. 17—TX-2 plug-in unit.

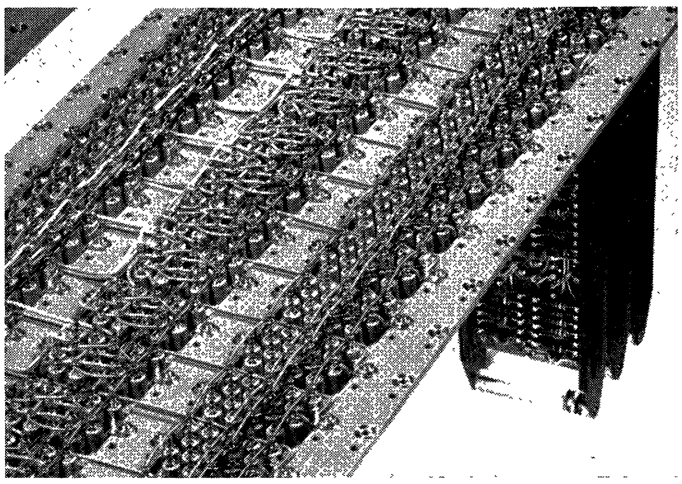


Fig. 18—TX-2 back panel.

Packaging

The number of types of plug-in units was kept small for ease of production and to keep the number of spares to a minimum. The circuits are built on dip soldered etched boards and the components are hand soldered to solid turret lugs. The boards are mounted in steel shells shown in Fig. 17 to keep the boards from flexing. The male and female contacts are machined and gold plated. The sockets are hand wired and soldered in panels as in Fig. 18.

Conclusion

The result of these design considerations is a 5-megapulse control and arithmetic element which will take less than 40 square feet of space and dissipate less than 800 watts of power. The simplicity of the circuits has encouraged a degree of logical sophistication which would not have been chanced before.

Acknowledgment

A number of people took part in the work reported here. Major contributions were made by B. M. Gurley, J. R. Fadiman, R. A. Hughes, K. H. Konkle, and M. E. Petersen.

Discussion

R. D. Gloor (Ramo-Wooldridge Corp.): What is the estimate of the expected mean-free-time between component failures for TX-2?

Mr. Olsen: The TX-0 Computer, which has been running eight hours a day since last April, has lost no transistors. So our experience with the TX-0 is that we expect the transistor portion of the machine to go for weeks without an error.

John Hayes (U.S.N.E.L.): What type of transistors are used in the flip-flops?

Mr. Olsen: The Philco Service Barrier Transistor was a key part of this development. It is tested to computer specifications.

We also use two or three thousand Micro-alloy transistors. We would like to use 100 per cent Micro-alloy transistors, but there were only two or three thousand available at the time we needed them. They have higher gains, particularly higher current, and appear to be much better transistors.

L. P. Retzinger (Litton): What is the propagation time per carry digit?

Mr. Olsen: About 40 millmicroseconds per digit. We made no effort to speed this up. This is a straightforward cascaded inverter, and it was the simplest type carrier we felt we could make. Even though it is slow compared to the rest of the circuits, in the over-all system it contributes very little to it in time or calculations.

Win Soule (Digital Techniques): How do you obtain visual indication of flip-flop position?

Mr. Olsen: We drive incandescent bulbs with a jumping transistor—a hardly satisfactory way of doing it: 400 transistors drive 400 incandescent bulbs. This is probably the best system as a whole, because it is not too expensive. We have been looking for less expensive ways for getting information.

L. H. Crandon (Autonetics): Are there any other sensitive parameters, different from voltage, which are used in marginal checking?

Mr. Olsen: One of course, can spend a lifetime comparing every parameter with every other parameter. Marginal checking gives you very good measure of most sensitive areas, and this is the one we concentrated on, and we feel that this is a reasonable approach to it, when one is limited by a limited length of time.

R. O. Barnes (Boeing): How much circuitry is represented in one plug-in unit (as shown in the figure) *i.e.*, how many flip-flops per unit?

Mr. Olsen: The figure shows that it contained one of the ten transistor flip-flops, plus three volume transistors. Three is in one package of cross section of one by two inches, one flip-flop plus a little logic; eight to twelve converters, or eight to twelve inter-followers.

Diagnostic Techniques Improve Reliability

M. GREMS†, R. K. SMITH†, AND W. STADLER†

Introduction

DIAGNOSTIC TECHNIQUES, as used in this paper, are aids for testing, sampling, and spot checking a computer program. These aids are employed to obtain evidence that the program is producing satisfactory answers. This evidence promotes confidence, assurance, and trust in the results. The reliability of the computing process is improved by the diagnostic techniques. Since the reliability of calculated results depends upon many phases of computing such as problem statement, coding, data presentation, and machine operation, diagnostic techniques also are concerned with the many phases of an integrated computing system.

The trend toward completely planned program-controlled diagnosis of errors can be illustrated by considering three levels of diagnostic techniques according to their degree of automaticity:

1) Manual techniques use program tests to cause a machine STOP in case of an error. Minimum planning is expended in the hope that a STOP will not occur.
2) Semiautomatic techniques prepare a logical pattern of program tests and record data before a machine STOP. Planning is required to prestore useful information before the occurrence of an error.
3) Automatic techniques require extensive planning, as part of a complete system, to circumvent a machine STOP. A programmed method is necessary to report trouble and still get results.

A most significant change in technique is shown by the jump from manual to semiautomatic methods. These methods include rules for storing pertinent data before performing a calculation or using a subroutine. If subsequent tests indicate the occurrence of an error, the data are available for print-out by an uncomplicated subroutine. This print-out of the trouble-report does not require special programming beyond the test for error.

The transition from manual through semiautomatic to the third level, automatic diagnostic techniques, demonstrates the steady and rapid growth of improved automatic computing methods. Change is necessary because the complexities of very large, selfcontained machine systems make it impractical to use manual diagnostic techniques. Imagine trying to analyze a storage dump of 32,768 words (which were machine-coded and machine-stored), in the hope of finding the source of an error. Instead, an automatic diagnostic routine, consisting of several techniques already in practice, is indicated. This type of diagnostic provides a record (or progress trail) of successfully completed stages of calculation and provides probable clues for finding trouble when an error is suspected. Also it enables some corrective action while calculations are continued at high speed.

It is obvious that preparation of an automatic diagnostic routine is expensive in planning time alone. However, the alternative of improved manual methods is incompatible with the published concepts of automatic programming. Advanced compilers which generate machine instructions or systems which plan storage allocation according to a hidden formula nullify the usefulness of such manual techniques as selective tracing or changed word post mortems.

Manual Techniques

Manual diagnostic techniques are not new to computing practices but are an outgrowth of the old familiar check-list. A check-list is used at progressive stages during hand computing to explain progress and to ensure that completed results are similar to anticipated results at each stage. The designers of computers recognize this need for a check-list, and include machine operation codes for conditional and unconditional STOPS and TRANSFERS; *e.g.* STOP, HALT, MS (Manually Selective Stop), EJ (Equality Jump) and TR O (Transfer on zero). The early computer programs would be unreliable without these operation codes and an accompanying check-list.

In a manual diagnostic routine the common practice for using conditional and unconditional STOP and TRANSFER codes is to include a STOP for any questionable situation. Then, when the STOP is executed, this manual procedure follows:

1) Manually copy all information from the console panel.
2) Dump or trace that portion of storage containing the STOP instruction.
3) Scrutinize carefully (on or off the machine) the instructions and the console information, hoping to isolate the error.
4) Decide on a corrective measure.
5) Employ the corrective measure in the program.

† Boeing Airplane Co., Seattle, Wash.

Manual diagnostic techniques are used almost exclusively in initial library subroutines. These library subroutines detect errors which cause the computer to stop. Then, the machine operator examines both the console information and a list of STOP explanations to isolate the error and decide on corrective action. The use of a check-list with machine operation codes is illustrated in one computer program where a specific angle is required. The sine of this angle is available from a previous computation in the program and the arcsine is to be computed by using a library subroutine. When the sine is greater than +1.0 or less than −1.0, something is wrong. Therefore, before the actual computing of the arcsine starts, a test is made of 1.0 minus the absolute value of the sine.

The regional instructions for the arcsine subroutine illustrate this test where X is the sine of the angle.

```
LOCATION   INSTRUCTION        EXPLANATION
   .
   .
   .
 F0318     R ADD  −E0004   −E0004 contains (1.0−|X|)
 F0319     TR+     F0321   Continue, |X| is ≤1.0
 F0320     STOP    F0100   Stop, |X| is too large
 F0321     R ADD    —      Continue computing
   .
   .
   .
```

Suppose that an incorrect sine value of 2.3715 is computed. This error is detected by not transferring on plus at instruction F0319. The execution of the next instruction stops the computer to indicate the error. The machine operator copies the contents of the instruction counter and the accumulator from the console panel to isolate the location of the error. He refers to a list of STOPS which include the following information for the arcsine subroutine:

REGIONAL LOCATION	DECIMAL LOCATION	OCTAL LOCATION	MEANING	CORRECTIVE ACTION
F0320	$t+66$	$t+102$	$\|X\|>1.0$ for arcsine or arccosine calculation.	Press START for second attempt. There is an error in your previous computation for X. Correct before rerunning.

The cause of error is decided and corrective action taken before the computation is continued. This method is wasteful of machine time and is frustrating to the programmer. It is a satisfactory method for short library subroutines, programs using limited storage, and hand-coded programs with little or no relocation of instructions.

Semiautomatic Techniques

Semiautomatic diagnostic techniques include not only the detection and isolation of errors, but also the recording of comments and pertinent data by the program. They are a direct outgrowth of the manual techniques and a stepping stone to a fully automatic routine. The recording of comments by the trail, when something is amiss, assures the engineer that unusual circumstances are recognized and therefore increases confidence in the reliability of results. This ability to isolate errors automatically and record pertinent information is a big step forward in diagnostic techniques. It assures that spot-checking is taking place during computing and is essential to computing and coding systems where relocation and storage assignment are delegated to the system itself. Currently, there are a number of successful diagnostic routines using these techniques for such computers as International Business Machines Models 701, 704, and 650; for Remington-Rand Models 1103A and UNIVAC; and for the Bendix G-15.

Semiautomatic techniques are used in the algebraic computing system, BACAIC.[1] When the BACAIC IBM 701 program detects an error, the program prints the reason for the error and suggests a corrective measure. It also prints a minimum of information to pinpoint the actual storage location of the error and the relative location of the computing in the problem.

The sample error print-out below shows the machine stopped at storage location 1051 while computing a square root in problem expression number six.

```
F108    SRT X MUST BE GREATER THAN OR EQUAL TO ZERO TO CONTINUE.   STOP.
            DECIMAL NUMBERS                          OCTAL NUMBERS
CONTROL PROG.  EXPRESSION NUMBER              STOPPED AT   TRANSFER TO
     31              6                           1051          4704
CASE RESULTS WRONG.  PUSH START FOR NEXT CASE, OR SENSE 1 FOR INTERRUPT
```

After printing this trail data, computer STOPS, as no corrective action is taken automatically by program.

When the BACAIC IBM 650 program detects an error, the program punches an error card and continues computing. This error card contains alphabetic and decimal information. The sample of an error card's contents given below shows that this data is sufficient to pin-point the error by expression and job case numbers:

[1] M. Grems and R. E. Porter, "A truly automatic computing system," *1956 Proc. Western Joint Computer Conf.*

	JOB	CASE	ROUTINE	LEFT OP.	RIGHT OPERAND	ERROR CODE	EXPRESSION NUMBER
ERROR	6015	0127	SRT	—	X−0.36000000 01	02	33

Where error code 02 is explained by:

2 ATTEMPTING TO FIND THE SQUARE ROOT OF A NEGATIVE NUMBER.

Three error cards are punched for one problem before the program automatically transfers to an interrupt routine which stops the computer.

Use of advanced diagnostic techniques prescribes that a large program be planned around a diagnostic routine. This requires an acute awareness of the fault areas in a problem. Sufficient checking should be included to ensure that when final results are computed they have a high degree of reliability.

Library subroutines, which are the backbone of a good programming system, should be consistent as a group for successful use in diagnosis. The control information for each library subroutine, which includes input and output data and exit instructions, should be in definite locations relative to the beginning of the subroutine, or else the locations must be specified in some manner. Consider the following instructions, which are the "front end" (beginning instructions) of each one of a set of library subroutines. These instructions are designed to work with a diagnostic routine provided by a compiler[2] written for the UNIVAC scientific computer.

```
t      MJ        0        START
t+1    TP       t+4       t+4
t+2    RJ     DIAG+3      DIAG
t+3    MJ        0        FILL
t+4     INPUT DATA BEGINS HERE
```

If the subroutine successfully completes its function a return is made through instruction ($t+3$), where FILL has been replaced with the address in the main program at which computing is to continue. If an error is found a return is made to the diagnostic through instruction ($t+2$), with the number of words of input data left in the accumulator. At instruction ($t+2$) control jumps to the diagnostic routine, and the address ($t+3$) is put into the fourth instruction of the diagnostic. The instruction ($t+1$) serves the important purpose of telling the diagnostic where the input data is located. With a consistent "front end" such as this on library subroutines the diagnostic operates more efficiently to point out errors that occur. It provides for printing of input data through the use of instruction ($t+1$), and indicates the location in the main program where the subroutine is used from instruction ($t+3$).

An example of a semiautomatic diagnostic routine gives some idea of the techniques used to help speed up production and increase the user's confidence in the results. When problems are a complex of many parts and computing is lengthy, a progress trail of intermediate results is valuable. It is inadvisable to compute for long periods of time with no indication of the progress of the problem. By looking at a trail of intermediate results one can tell when a problem goes astray. Also when final results are obtained they have a higher probability of being reliable if a checking of the trail shows little or no deviations from the expected values. In the event of an error the diagnostic indicates which part of the program is operating at that time. In most cases it suggests possible alternatives to use to continue computing, and frequently it restarts the problem from some point at which correct results are known.

The following diagnostic is used in an airplane and target system simulation programmed for the IBM 701. The problem involves very little input data and a long period of computing. The diagnostic is written to comment on the following three types of conditions:

1) Changes in the flow of the program by printing a trail of the progress of the problem.
2) Errors found by subroutines in their input data.
3) Errors found as a result of checking the arithmetic and logic operations of the main program.

Before computing begins, all initial input data with headings and comments are printed as part of the trail. These data then become immediately available if checking through the trail is necessary, and may show erroneous values not found by the data read-in program. The progress trail is a printing of informative data at selected points in the sequence of operations. The point where printing occurs is determined by the particular part of the program in operation and by a flight plan read in as part of the input. The following sample of a print-out is typical of trail information.

AIRPLANE AND TARGET POSITION

TIME	X AIR	Y AIR	H AIR	X TARGT	Y TARGT	H TARGT
1.93	3861.63	730.24	2341.55	5773.21	2604.67	1817.20[3]

Each comment and its associated values represents an entry into the diagnostic. The comment and particular values to print are determined by a code left in the MQ register for the diagnostic. Control is turned over to the diagnostic which interprets the code and prints the

[2] M. B. Lieberknecht, L. J. McPhee, M. Morris, W. K. McKinley, and R. E. Porter, "The Boeing programming system for the UNIVAC scientific computer model 1103A." In preparation.

[3] These values are fictitious.

necessary information. After printing, control reverts to the main program, and computing continues.

The values printed are used particularly in checking the flow of the problem. For example, when the printed values of airplane position at time (t) differ from the expected values by a large factor, then the operator knows that the problem is not progressing successfully. Stopping the machine at this point could save considerable computing time, particularly if the problem is to continue computing for a long period after this.

As computing progresses many library subroutines are used. The subroutines are written with consistent entry and exit points, and are used as part of the diagnostic. They check their input data to determine if it is compatible with the requirements of the subroutine. If a discrepancy occurs, a code denoting the type of error and the program number are placed in the accumulator and the subroutine transfers to an error exit. All the subroutines use a common input-output region, called the J region. Just prior to using a subroutine input data is stored in the J region, and when the subroutine is finished it stores its output in the J region. At various times during the computing, information is stored in a control region (called the X region). This information includes a code denoting the particular part of the program in operation, transfer locations, and various operational values. Both the J region and the X region are available to the diagnostic at all times. A sample of the information printed when an error is found in the input to a subroutine is described below.

```
X IS TOO LARGE PROG. 6203 R ADDR 1726 TBL 20
 P-C-S    TR ADDR    TIME     INPUT     INPUT
 1-3-1     1742       863    7369201
```

PROG. 6203 identifies the subroutine in which the error occurred. This particular subroutine is a table look-up and interpolation subroutine. The print-out indicates that the argument value exceeds the argument values in the table. The R ADDR indicates the location in the main program where entry was made to the subroutine. The R ADDR along with the INPUT are found in the J region while the other values are found in the X region. TBL gives the number of the last table used or the table being used at this time; P-C-S refers to the particular part of the main program in operation at the time of error; TR ADDR is an address where control may be transferred in order to modify the computing sequence and either continue computing or prepare to remove the problem from the machine; TIME is airplane flight time; INPUT refers to the argument or arguments used by the subroutines. Following the error print-out, the machine stops. The decision as to what should be done at this point is left to the operator.

Checking for arithmetic and logic errors is done by the main computing programs. Depending upon the type of error, values from the X region may or may not be printed. When data are printed along with the comment, the print-out is essentially the same as that for subroutine errors. In the case where no values from the X region are printed with the comment a statement is usually included which suggests some action to perform. The latter case is illustrated by sample printings which follow:

```
CANNOT LOCATE TABLE ON 257. START TO REWIND
ITERATION COUNT EQUALS 20
```

After each error comment the machine stops. The first comment indicates that by restarting the machine, tape 257 will be rewound and a second search made for the particular table. Some comments, such as the second comment above, do not indicate the next step, and after the machine stops the operator decides what course of action to follow.

It is noted that after printing an error comment any decision made is left to the operator. It is now felt that many of the decisions which are defined for the operator could be included in the programming. In a number of instances this allows the machine to continue computing without stopping, and thereby saves considerable time.

Each of the parts which make up the main computing programs are checked out (debugged) individually before being incorporated in the complete program. However, the true test of their correctness comes when the combined parts are checked out as a unit with sample problems. The diagnostic serves one of its most important functions at this time by giving more concrete information as to the type and location of programming errors. Also, it has an indirect application in that it helps give better clues to the service engineers for locating machine malfunctions.

Automatic Techniques

The preceding example illustrates that the second level, semiautomatic diagnostic routine leaves a trail of problem progress. Reliability is promoted thereby because rather permanent evidence is available which shows the actual path of the calculation. Furthermore, intermediate answers allow hand-calculated checks upon the methods or the final results at any convenient time when added assurance is felt necessary. However, the routine in the example is termed semiautomatic because analysis of the trail and any corrective actions are performed at manual speeds after a machine STOP. Any such manual procedure is subject to fumbles, errors, and costly delays even for small programs. More advanced, program-controlled techniques are necessary to improve reliability as problems become very large and thereby multiply the costs of manual machine procedures. The confusing mass of detail concealed in an integrated computing system prohibits any person or group of people from tracing trouble in the earlier fashion of STOP, copy control console, consult check-list, consider locations in storage, dump large portions of storage, and then retire gracefully for analysis. These slow procedures impede communication between man and machine and are contrary to more sophisticated thinking as expressed in the deluge of literature on "automatic programming."

Of course, progress has been made in reducing the communication difficulties in transferring a problem to a machine. For example, the compiler[2] for the 1103A accepts several kinds of language including algebraic symbols, such as those in Fig. 1, which are quite familiar to the engineer. A problem consisting of 88 expressions including those three in the figure were recently prepared for the Boeing Airplane Company Algebraic Interpretive Computing[1] system (for an IBM 701) in a few hours. The significant fact is that the problem statement *precisely as it goes into the computer* is familiar and understandable to the originator of the problem who has neither time nor inclination to learn about subroutines, machine language, or diagnostic techniques. But, while communication *to* machines has been effected and documented, communication *from* machines has not been equally aided or popularized.

Expressions from Cloudy Ball Calculation

ALG′ $-$ SIN X/COS $X=H$

ALG′ (COS $X-$SIN X/X)/(SIN $X+$COS X/X) $=I$

ALG′ (3.0 COS $X/X-$(3.0/$X/X-$1.0) SIN X)/(3.0 SIN X/X +(3.0/$X/X-$1.0) COS X) $=J$

Fig. 1—Example of communication to the Boeing 1103A compiler.

A diagnostic routine which provides a great deal of communication *from* an IBM Model 701 will be illustrated. The diagnostic duties are performed by a diagnostic print routine and by special programming in a sample problem. This one-program diagnostic is written to prove that automatic techniques are possible. It provides for general and specific handling of error situations. The general error is reported through a standard entry, from a conventional subroutine, to a diagnostic print routine. This error report is made with minimum interference from the main program. Specific error techniques are programmed as part of the main program and explained (at compute time) by the diagnostic print routine. Even if there is no error, periodic progress reports are set up by the main program and printed by the diagnostic.

In this sample problem—computing, diagnosis, error correction, progress printing, and input-output are performed by a combination of three routines. Fig. 2 shows the three classes of routines used: main program, diagnostic print, and standard subroutines. Each class of routine has access to common storage. The main program sets control data in storage, links to subroutines, performs tests, and links to diagnostic print. The main program is the only means of performing special error activity like replacing bad data.

The diagnostic print performs all of the program printing including input, progress trail, error comments, and output. The diagnostic will print from a general error code left by a subroutine or will respond to a special error code from the main program. In either case, the error report is prefaced by a block title line which indicates what part of the main program is in operation. That information is available at error time because it had been prestored in the common storage region.

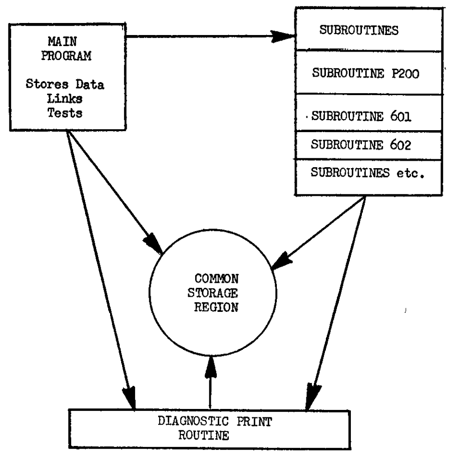


Fig. 2—Three classes of routines for a sample problem.

Fig. 3 (opposite) shows the various entry points to the diagnostic print routine. Particular interest should be directed at the entry number 8 which provides for error-explanation printing without special programming. In contrast, diagnostic print entry 6 is used after a special comment code has been stored by the main program. It implies that further special programming for the error is provided within the main program. This additional programming is effective before or after the trouble print-out with the intention of continuing the normal program as soon as possible.

Common storage is used by the main program and by the diagnostic print; subroutines also use common storage to find their control data. If an error occurs, this information in common storage is readily available without extensive searching. Fig. 4 indicates the kind of information placed in the common storage region. Locations reserved for Accumulator and MQ values, in common storage, does not imply that those values are prestored. It is merely a convenient arrangement for print-out of that information at error time.

Exactly what data must be prestored is ruled by subroutine conventions which hold for this sample problem. Besides specifying which data must be prestored, the rules provide a basic linkage, and prescribe information which must be in the accumulator and in the MQ for an "error return." (See Fig. 5.) These three conventions illustrate the type of consistency which is necessary for any automatic system.

The parts of a diagnostic routine and main program for a particular problem are described in Figs. 2–4. A list of instructions from part of the main program (refer to Fig. 6) illustrates the use of a diagnostic. The F1400 block in Fig. 6 has two instructions which cause entry to the diagnostic print routine. Only the instruction at F1419 (TR D0005) is executed each time the program is used. This entry to the diagnostic will cause

1) Page headings.
   Uses common storage locations C, D, and E.
2) Column headings.
   Uses common storage location F.
3) Print data.
   Uses common storage locations G, H, and I.
4. Print comment.
   Uses common storage location F.
5) Print progress trail.
   Uses common storage locations A and B.
6) Error print.
   Uses common storage locations B, F, L, M, N, O, P, Q, R, S, etc.
7) Error print with erasable storage.
   Uses common storage locations B, F, J, K, L, M, N, O, P, Q, R, S, etc.
8) Print from general subroutine error code.
   Use success return and common storage locations B, F, L, M, N, O, P, Q, R, S.
9) Hardware test.
   Loads and uses test routine from magnetic tape.
10) Open ended for more entries to diagnostic routine.

(See Fig. 4 for locations in common storage referred to by letters A–S.)

Fig. 3—Entry points to the diagnostic print.

A—Code number for major block title.
B—Code number for minor block title.
C—First code number for page heading.
D—Second number for page heading.
E—Page count.
F—Comment code number, drum or tape.
G—Data location.
H—Location of data index.
I—Amount of data.
J—Location of erasable storage.
K—Size of erasable storage.
L—Contents of ACCumulator left half.
M—Contents of ACCumulator right half.
N—Contents of *MQ* left half.
O—Contents of *MQ* right half.
P—Subroutine code number.
Q—Amount of control data.
R—Control data 1.
S—Control data continued.

Fig. 4—Contents of common storage.

a print of the trail of progress. The trail consists of the major and minor block titles which are stored and printed periodically.

The TRANSFER instruction at F1412 (TR D0008) is at the "error return" position (in the main program) which is used only if trouble is encountered by a subroutine. In the case of the error return and TRANSFER at instruction F1412 the diagnostic print routine would operate as follows:

1) The contents of the Accumulator and MQ are stored in the common region.
2) The block title code, prestored as a code number in common storage location *B*, is printed:

TEST *T* AND *Q* FOR LIMITING CONDITIONS

1) All subroutines are entered by basic linkage as follows:

Main Program Locations			*Explanation*
r	*R* ADD	r	The address r is in ACCumulator.
$r+1$	TR	t	Transfer to subroutine.
$r+2$	TR		Subroutine returns here if error.
$r+3$	TR		Subroutine returns here if success.

2) Before using a subroutine the following information must be placed in common storage:

   Minor block title code in location B----.
   Subroutine code name in location P----.
   Control data as required by the subroutine in Q----, R----, and S----.

3) All error returns from subroutines will be to ($r+2$) of the main program and must have a general error code in the ACCumulator and the success address in the *MQ*. (There are 57 general error codes so far, but the list is open ended.)

Fig. 5—Subroutine conventions.

```
F1400  R ADD   Z0022 }  Set block code title.
F1401  STORE   B----
F1402  R ADD   X0007    Set subroutine code name.
F1403  STORE   P----
F1404  R ADD   B0002    Constant 2—amount data.
F1405  STORE   Q----
F1406  R ADD   Cxxxx
F1407  STORE   R----    First five pieces of data
F1408  R ADD   Exxxx      are stored in common
F1409  STORE   S----  }   storage (see Fig. 4).
F1410  R ADD   F1410    Basic linkage to subroutine
F1411  TR                 —to test Y_i and q_t
F1412  TR      D0008    If error return, TR to diag-
                          nostic.
F1413  NO OP            Success return.
F1414  R ADD   Z0023
F1415  STORE   A----    Major
F1416  R ADD   Z0024      and minor block titles.
F1417  STORE   B----    Stored in common storage.
F1418  R ADD   F1418    Basic linkage to
F1419  TR      D0005      diagnostic entry 5.
```

Fig. 6—Sample of main program instructions.

3) The contents of common storage locations *L* through *S* are printed:

```
   ACC            MQ        SUB   CNTRL  DATA  DATA
44     0   1734         0   603     2    1201   36
```

4) The explanation of error code 44 (found using the code from the Accumulator) is printed:

IMPROBABLE ZERO DATA—AUTOMATIC TRANSFER TO TEST HARDWARE

5) Most general error codes allow a success return to the main program (in this case to the actual address (1734) which was obtained from the MQ). However, error code 44 is one of several malfunction codes which causes an entry to the diagnostic hardware test. The hardware test is a machine testing routine obtained from magnetic tape. This routine is loaded and used by means of a transfer to the entry of the diagnostic.

One other error return is programmed in the F1400 block of the sample problem. (Refer to Fig. 7.) The

```
F1420   R ADD   Z0025    Block title code for
F1421   STORE   B----      calculation of V.
F1422   R ADD   X0008    Subroutine code name.
F1423   STORE   P----
F1424   R ADD   B0002    Amount of control data.
F1425   STORE   Q----
F1426   R ADD   Exxxx    Control data.
F1427   STORE   R----
F1428   R ADD   Cxxxx    These five pieces of data
F1429   STORE   S----      are placed in common
                           storage.
F1430   R ADD   F1430    Basic linkage to subroutine.
F1431   TR                 —to calculate V=sum z_ij
F1432   TR      D0008    Error transfer to diagnostic.
F1433                    Success return from sub-
          .                routine.
          .              Continue computing.
          .
```

Fig. 7—Sample instructions from main program.

TRANSFER to D0008 (instruction F1432) is executed upon an error return from a subroutine which performs a certain calculation of V. Entry D0008 of the diagnostic print results in the following printed record:

```
CALCULATION OF V EQUALS SUM OF Z, IJ
     ACC              MQ         SUB  CNTRL  DATA  DATA
53        0  1754           1999  605      2   3201    36
THE CALCULATED RESULTS ARE BEYOND NORMAL
RANGE
```

In this case control is returned to the success return in the main program which is the actual address (1754). The trouble has been recorded but calculation continues on the assumption that the results will be of value. This is an automatic diagnostic technique.

Conclusion

In conclusion, programmed reliability using automatic (nonstop) diagnostic techniques is possible. This fact is demonstrated by examples of successful semiautomatic approaches which are performing in a nearly automatic manner. The amount of programming foresight necessary to provide for probable errors is illustrated by the sample problem and diagnostic routine outlined in Figs. 2 through 7. Experience with several computers indicates that the techniques and planning for automatic diagnostics are the same for all machines although the mechanics of their operation are different. Advanced techniques are deemed essential for reliable problem solution by very complex systems. The advantages of machine selfcoding, machine storage planning, and machine selfprogramming are cancelled by human debugging. Only nonstop, program-controlled recording and correcting measures can operate with the thoroughness necessary to ensure the success of automatic machine methods.

Discussion

John Paivinen (General Electric): What is the ratio of diagnostic instructions to main program instructions?

Mr. Stadler: About one to three.

R. C. Boden (IBM): What percentage of running time is used by your programmed checking routines when the machine is operating correctly?

Mr. Stadler: The percentage of running time is best calculated by the time it takes to print out. The computing is insignificant.

M. I. Bernstein (The RAND Corp.): What percentage of debugging time is spent debugging that part of the code which uses the diagnostic routines?

Mr. Stadler: The diagnostic routines do require some debugging time, but they are in the form of subroutines, and subroutines are checked separately and represent a small amount of check-out time. Subroutines in the example of airplanes and target systems simulated in the diagnostic routines, was probably about a week. This diagnostic routine helps debugging of the rest of the program appreciably, and helps maintenance engineers find machine trouble as well.

Error Detection and Error Correction in Real-Time Digital Computers*

ANTHONY RALSTON†

Introduction

UNTIL the digital computer is built which never malfunctions, programmers will have to worry about what will happen to their programs if a machine error does occur. In the case of computers used as integral parts of real-time control systems "worry" is perhaps too weak a word. For a machine error in a computer in such a system may not just cause trouble; it may cause disaster. Thus, with the increasing use of digital computers as elements of real-time systems it has become increasingly important that techniques be developed to handle the malfunction problem in real-time computers. The purpose of this paper is to present a number of such techniques—programming techniques—some well-known and some new for the detection and correction of performance errors in real-time digital computers.

The key word in the previous sentence, is of course, "correction." As long as digital computers have been used for non-real-time applications, whether scientific or business, methods to detect machine errors have been a basic part of the programming effort. To correct such errors however, the usual technique has been, upon detection of the machine error, to stop the computer and rerun the problem or at least to rerun the problem from the point of the last successful programmed check. In a real-time digital computer such a procedure is clearly not feasible. First of all it is quite obvious that when a computer is a link in a control system it just cannot be stopped without bringing the system to a grinding halt (or perhaps something much worse). This is equally true whether the computer is controlling a reaction in a chemical plant or is being used to fire an antiaircraft gun. Secondly, owing to the real-time nature of the computer it is not possible to repeat more than a *very small portion* of the computation when a malfunction is detected. Thus, when a malfunction is detected there must be as part of the over-all program a routine which, in a *very small amount of time*, "corrects" this machine error. In this paper the term "correction" will be used both in its usual sense to mean obtaining the true value of the quantity in error and to mean obtaining a sufficiently close approximation to the true value to enable the computation to proceed. Also, when the malfunction causes not a direct arithmetic error but rather some logical error, "correction" will be used to mean the repair, exact or approximate, of such a machine error.

* This work was done at the Bell Telephone Labs., Whippany, N. J., under Air Force Contract AF33(600)-21536.
† Bell Telephone Labs., Whippany, N. J.

Although we have thus far emphasized correction of machine errors, the detection of such errors in a real time situation has problems connected with it which are not present in the non-real-time case. In the first place the exigencies of the real-time case are such that the detection process must take up only a small percentage of the total computation time since we will be looking for errors many times for each time we actually find an error. Therefore, for example, performing the whole computation twice and periodically comparing results is a method which in general cannot be tolerated.[1] Furthermore, since the detection of an error is always followed by correction, detection processes should be used which facilitate as much as possible the correction process if a malfunction is detected. Thus in what follows we will generally consider each detection technique to be directly related to a correction technique.

Before proceeding further we should make it clear that the methods discussed here are aimed at detecting and correcting random, transient malfunctions (*e.g.*, dropped bits, incorrect execution of an instruction, incorrect reading of a register in memory). Therefore, the term malfunction (or machine error) from here on will have this meaning and will exclude outright equipment breakdowns. In particular most of the methods to be discussed are aimed at detecting and correcting single isolated transient machine errors (*e.g.* a single dropped bit) although some types of multiple or recurring malfunctions (*e.g.*, more than one dropped bit) are also detected and corrected. (Some of the methods will also detect and correct certain malfunctions external to the computer which might, for example, cause incorrect data to be sent to the computer.) That this is a reasonable way of attacking the error detection and correction problem is borne out by various studies, among them one made at the Bell Telephone Laboratories on the Tradic Computer that indicated a ratio of greater than three-to-one of single machine errors to multiple machine errors. We note here the obvious fact that a single malfunction such as a dropped bit may cause a multiple-bit error in a later computed quantity. Our methods are aimed at detecting and correcting single malfunctions even though such an error may later cause a multiple-bit error in a computed quantity. In what follows the term "error" when used alone is meant to signify the result of a malfunction.

[1] Of course, if a double computation check is performed by two computers operating in parallel then there is no loss of time in performing this check. However, even if the economics of the situation permit this type of operation, physical restraints on the weight and size of the computer often will make it impossible.

Classification of Machine Errors

The most important factor in deciding what kind of detection and correction process to use in a given situation is how exact the arithmetic quantity (or logical process) in question must be after correction in order for the system to continue in reasonable operation. It is clear that exact correction will require more time than that required for performing a computation twice and then comparing the results, since this will detect but not correct errors. Therefore, time limitations will make it important to decide when approximate corrections are enough to keep the system in reasonable operation. In this connection we may classify errors as follows:

1) Vital errors—These are errors which if not corrected exactly (or to some specified number or significant bits)[2] may cause complete breakdown or failure of the system.
2) Serious errors—These are errors which will cause the system to fail only if they are repeated sufficiently often. In general approximate correction of such errors is enough to keep the system operating properly.
3) Nonserious errors—These are errors which will be smoothed out in time (unless repeated at very high frequency) and thus can usually be neglected.

These classifications have been kept purposely very general so as to try and cover the field in a simple fashion. There are of course gradations within each category and between the categories. In the example of a computer in a missile guiding the missile to a target, a vital error would be the loss by the computer of the position of the missile (if there were no external means of recovering the position); a serious error would be an incorrect computation of an increment to the present position (which, if the increments were small, would not cause a vital error in the end result unless the error was repeated often); and a nonserious error would be a minor error in the steering order to the missile at an early stage of the flight.

In discussing methods of detection and correction of errors it will be convenient to classify the types of errors somewhat differently than above as follows:

1) Arithmetic errors—These are errors whose only effect is to cause a computed quantity to be incorrect.
2) Sequencing errors—These are errors which cause the program to get out of its proper sequence. Sequencing errors will, in turn, of course, cause arithmetic errors.

Again these categories must not be considered rigid. For example, it is clear that certain errors in computed quantities may at some later time cause the program to get out of its proper sequence. Such an error would be difficult to classify as above. But these categories will serve as a starting point in our discussion of methods of error detection and correction.

Techniques for the Detection and Correction of Arithmetic Errors

By Multiple Storage and Computation

We have previously said that detection of errors by computing a quantity twice is in general too time-consuming for a real-time system and we have also noted that this method will detect but not correct errors. However, in the case where an uncorrected error would be vital we generally have to detect and correct exactly even if the process we use is time-consuming. (Luckily of course few types of errors fall into the vital category.) One way of avoiding vital errors is to use the well-known method of computing and storing the vital quantities at least three times. As an example let us assume that we have a vital quantity Q (*e.g.*, the present position of the missile in the previous example) which is being regularly changed by adding an increment ΔQ to it. Let us assume that errors in ΔQ are only "serious" so that even if ΔQ is in error $Q+\Delta Q$ will still be sufficiently accurate to let the system operate. This is the same as saying that only the first k significant bits of Q are vital and that we have constrained ΔQ to be so small that it cannot (in one addition) affect these k bits except by a carry. Then our problem is to safeguard these k significant bits when holding Q in storage and when adding ΔQ to Q. Let Q be stored in registers a_1, a_2, and a_3 and let ΔQ be stored in registers b_1, b_2, and b_3.

We assume that: 1) At most one of the six quantities in a_1, a_2, a_3, b_1, b_2, and b_3 is incorrect and the computation about to be described is performed correctly or, 2) all six of the quantities are correct and at most one arithmetic error occurs in the computation about to be discussed.[3]

That is, we assume that not more than one malfunction directly affecting Q occurs per recomputation of Q. Then a possible sequence of operations (though not necessarily the most economical in program steps) which will compute (the first k significant figures of) $Q+\Delta Q$ correctly and place the result correctly in a_1, a_2, and a_3 is:

$$\left.\begin{array}{lll} 1) & \text{Clear and add} & a_1 \\ 2) & \text{Add} & b_1 \\ 3) & \text{Store in} & a_1 \\ 4) & \text{Clear and add} & a_2 \\ 5) & \text{Add} & b_2 \\ 6) & \text{Store in} & a_2 \\ 7) & \text{Clear and add} & a_3 \\ 8) & \text{Add} & b_3 \\ 9) & \text{Store in} & a_3 \end{array}\right\}\text{Puts } Q+\Delta Q \text{ in } a_1, a_2, a_3$$

[2] In this paper we will use the language of binary computers but most of what is said holds equally well for computers using other number systems.

[3] By "correct" for ΔQ we mean that the magnitude of ΔQ is such that it cannot affect the first k significant bits of Q, except by a carry. Clearly the case discussed here includes the special case in which every bit of Q is vital and where, therefore, ΔQ must be exactly correct.

```
10) Clear and add $a_1$                 } Tests to see if $C(a_1)$
11) Subtract      $a_2$                 }   $= C(a_2)$[4]
12) Transfer if nonzero to 17           }
13) Clear and add $a_3$                 } Tests to see if $C(a_2)$
14) Subtract      $a_2$                 }   $= C(a_3)$
15) Transfer if nonzero to 21           }
16) Transfer out                          End of routine
17) Clear and add $a_3$ }
18) Store in      $a_1$ } If $C(a_1) \neq C(a_2)$ replace both by
19) Store in      $a_2$ }   $C(a_3)$
20) Transfer to   16    }
21) Clear and add $a_2$ }
22) Store in      $a_3$ } If $C(a_3) \neq C(a_2)$ replace $C(a_3)$ by
23) Transfer to   16    }   $C(a_2)$
```

An analysis of this routine will easily indicate that if condition 1) or 2) is satisfied, at the end of the routine a_1, a_2, and a_3 will contain $Q+\Delta Q$ correct to k significant bits. That is, any single malfunction affecting the storage of the quantities or the computation of $Q+\Delta Q$ will have been detected and corrected. The routine also detects and corrects certain multiple errors (*e.g.*, two separate malfunctions involving a_1). One way to simplify the routine with little loss in safety is to make $C(a_3)=C(a_2)$ if $C(a_1)=C(a_2)$. To do this we eliminate steps 13)–15) and insert steps 21)–22) after step 12). On the other hand, of course, more sophisticated routines using quadruple or higher order computation and storage will detect and correct more general multiple errors.

Multiple storage and computation may also be used to protect nonphysical vital quantities. An example of such a quantity might be a count which if incorrectly stored or tested might cause a vital error in some associated physical quantity or a vital sequencing error. This count could then be triply stored and triply tested.

By Extrapolative Checking

This method is a type of reasonableness check[1] and should have wide application because of its computational simplicity and high effectiveness in detection and correction of serious errors. The essence of the method is to compare the value of a quantity as computed directly from physical data with the value found by extrapolating from previous values. Then using the known error bounds and physical bounds in the system, gross errors may be detected and corrected[5] although small errors may not be detected. We will consider two mathematical formulations of this method, the first in which we use direct Lagrangian extrapolation from previous values and the second in which we introduce smoothing before extrapolating. Before discussing these methods we will define some notation. Let $y(t)$ be the quantity we are computing. For example $y(t)$ might be the ΔQ of the previous example. We will assume we are computing the function at equal intervals of time. Let

y_i^T be the true physical value of $y(t)$ at time t_i
y_i^C be the value of $y(t)$ at t_i correctly computed from physical data
y_i^E be the extrapolated value of $y(t)$ at t_i

and define

$$h = t_{i+1} - t_i$$

and

$$\epsilon_i = y_i^C - y_i^T \qquad (1)$$

and finally let $M=\text{Max}_i|\epsilon_i|$.

Lagrangian Extrapolation: We will discuss in detail the case where three previous values of $y(t)$ are used in the extrapolation. The discussion for any other number of points is completely analogous. Using the Lagrangian interpolation formula [2] to find the extrapolated value of $y(t)$ at time t_3 from the values at times t_2, t_1, and t_0 we get

$$y_3^E = 3y_2^C - 3y_1^C + y_0^C \qquad (2)$$

where the true values of $y(t)$ at these four points are related by the formula

$$y_3^T = 3y_2^T - 3y_1^T + y_0^T + h^3 y'''(\xi) \qquad (3)$$

where the primes indicate differentiation[6] and $t_0 \leq \xi \leq t_3$. We are interested in comparing y_3^E with y_3^C, so we compute using (1), (2), and (3)

$$\begin{aligned} y_3^E &- y_3^C \\ &= 3y_2^C - 3y_1^C + y_0^C - y_3^C \\ &= 3y_2^T - 3y_1^T + y_0^T - y_3^T + 3\epsilon_2 - 3\epsilon_1 + \epsilon_0 - \epsilon_3 \\ &= \epsilon_0 - 3\epsilon_1 + 3\epsilon_2 - \epsilon_3 - h^3 y'''(\xi) \qquad (4) \end{aligned}$$

thus we have

$$|y_3^E - y_3^C| \leq 8M + h^3 \underset{[t_0,t_3]}{\text{Max}} |y'''(\xi)|. \qquad (5)$$

Both M and the maximum of the derivative can usually be estimated from a knowledge of the physical system and computational procedures being employed. Thus the bound on $|y_3^E - y_3^C|$ can be estimated quite accurately.

This bound is in general quite conservative but can be used to insure that no errors greater than the bound given by (5) occur. To use this method on the computer then, we would first compute y_3^C, then find y_3^E and compare the two. If the difference was less than the bound or "gate" we have chosen we would use y_3^C as the value of $y(t)$ at t_3. If not we would use y_3^E. We note here that it may be necessary to do the first few steps of the computation without extrapolative checking since in general we will have no past values of the function at the start. However it may be possible to provide some pseudo-past values of the function to use at the start. These pseudo-past values would generally be derived from a knowledge of how the physical system would behave

[4] $C(a_1)$ = Contents of a_1.
[5] Gross errors not only in the computer but also in the input to the computer will be detected and corrected.
[6] We assume here and in all that follows that functions we consider have derivatives of as high orders as required in our formulas.

initially and would be such that good agreement would be expected between the computed values and the values extrapolated from the pseudo-past data.

Smoothed Extrapolation: Again here we will discuss the method when the three points y_2^C, y_1^C and y_0^C are used for smoothing, but as before the discussion for more points is completely analogous. In particular we will use these three points to determine a first degree least-squares polynomial. Then we will use this polynomial to calculate y_3^C. This first degree polynomial $p(t)$ (or any higher degree polynomial when more points are being used in the smoothing) may be easily calculated using the Gram polynomials[7] and is

$$p(t) = 1/6[5y_0 + 2y_1 - y_2] + (1/2h)[y_2 - y_0]t \quad (6)$$

where we have let $t_0 = 0$ for convenience. The value of $p(t)$ at $t = 3h$ gives us the extrapolated value of $y(t)$ at t_3 and is

$$y_3^E = 1/3[4y_2^C + y_1^C - 2y_0^C]. \quad (7)$$

In terms of the true values of $y(t)$ at these four points the exact equation is

$$y_3^T = 1/3[4y_2^T + y_1^T - 2y_0^T] + h^3y'''(\xi) + 5/3\delta^2y_1^T \quad (8)$$

where $\delta^2y_1^T = y_0^T - 2y_1^T + y_2^T$ is the second central difference of $y(t)$ taken about t_1. This part of the error term is found by evaluating the first dropped Gram polynomial term in (6) at $t = 3h$.[8]

Now using (1), (7), and (8) we get, analogously to (4),

$$y_3^E - y_3^C = 1/3[-2\epsilon_0 + \epsilon_1 + 4\epsilon_2 - 3\epsilon_3] - h^3y'''(\xi) - 5/3\delta^2y_1^T. \quad (9)$$

Therefore

$$|y_3^E - y_3^C| \le \frac{10}{3}M + h^3 \operatorname*{Max}_{[t_0,t_3]} |y'''(\xi)| + \frac{5}{3}\operatorname*{Max}_{[t_0,t_2]} |\delta^2y_1^T|. \quad (10)$$

The basic differences between (5) and (10) are as we would expect. That is, since M is essentially a measure of the noise—both physical and computational—in the system, the smoothed extrapolation tends to reduce the error due to this noise, the maximum errors being in the ratio 10:24. On the other hand the function $y(t)$ is a physical quantity and thus the first degree smoothing process does not extrapolate it as accurately as the second-degree Lagrangian process. The added error, $5/3 \text{ Max } |\delta^2y_1^T|$ may be estimated by $5/3\, h^2 \text{ Max } |y''(\eta)|$ for η in $[t_0, t_2]$.[9] As in the case of Lagrangian extrapolation, (10) may be used to set a gate size to test whether the computed value, y_3^C, lies sufficiently close to y_3^E. It should be emphasized that the setting of this gate size in either case is an extremely important and basic part of using extrapolative checking. It is beyond the scope of this paper to go deeply into this problem. However, it is clear that a thorough knowledge of those components of the system which are providing inputs to the computer and a thorough understanding of the numerical process within the computer itself are required before M and the derivatives of $y(t)$ can be estimated properly. Then the setting of the gate size is a matter of determining from a knowledge of the error distributions, whether a less conservative gate size than the maximum error bounds of (5) and (10) is desirable. A gate size smaller than the maximum error bound might be chosen if values of $|y_3^E - y_3^C|$ close to this bound were considered extremely unlikely. If a gate size smaller than the maximum error bound given by (5) or (10) is chosen, then some correctly computed values of $y(t)$ will be detected as errors and the extrapolated value will be used. However, the maximum possible difference between the value of $y(t)$ that is used and the correct computed value will still be given by (5) or (10).

It is worthwhile to note that it may be desirable or necessary to change the gate sizes occasionally during the course of the computation. This change might be required during periods of particularly rapid change of physical quantities. Or perhaps estimates of M from (5) or (10) would make a change in gate size seem desirable.

The choice of whether to use Lagrangian or smoothed extrapolation again requires a thorough knowledge of the system. If noise is the predominant type of error expected, naturally smoothed extrapolation would be used. However, if the physical quantities involved are varying rapidly, Lagrangian extrapolation would probably be a better choice.

One question we have thus far neglected is: How can we insure that the extrapolated value itself is not badly in error? A simple way of checking the extrapolated value is to extrapolate a second time using a different formula and compare the results. For example, in the Lagrangian case, we might use the two-point formula

$$y_3^{E'} = 2y_2^C - y_1^C \quad (11)$$

where, using the error term related to (11), we may compute

$$|y_3^E - y_3^{E'}| \le 4M + h^3 \text{ Max } |y'''(\xi)| + h^2 \text{ Max } |y''(\eta)| \quad (12)$$

where ξ is in $[t_0, t_3]$ and η is in $[t_0, t_2]$. Then testing the two extrapolations against a second gate we would either accept the first extrapolation as correct or, if the difference in the two extrapolations exceeds the gate, we would use the computed value without a check. This assumes as before that no more than one malfunction has occurred in the computation of y_3^C and the related check.

The above procedure for checking y_3^E guards against gross errors in y_3^E caused either by a malfunction during

[7] See Hildebrand [2], p. 287.
[8] *Ibid.*, p. 294.
[9] See Kopal [3], pp. 102 ff.

the computation of y_3^E or by a mutilation in memory of the past values of $y(t)$ used in the extrapolation. The latter cause of error is protected against owing to the fact that the coefficients of y_0^C, y_1^C and y_2^C are different in (2) and (11). Another possible way of checking the extrapolation would be to use y_3^E, y_2^C and y_1^C to "back extrapolate" for y_0^C using (2). This method, however, would not detect mutilation in memory.

Still another question which must be answered is: how is the extrapolation at the $n+1$st step affected if at the nth step the extrapolated, rather than the computed, value was used due to a malfunction? If, for example, we put y_2^E instead of y_2^C in (2) and (11) and calculate the changes in (5) and (12) caused by this, we would see that the gate sizes used at t_3 would have to be increased. Indeed it is true in general that, if an extrapolated rather than a computed value is used at one step of the calculation, then at the next step the gate sizes must be increased. These increased gate sizes must be used in succeeding steps until the first extrapolated value no longer affects the computation of later extrapolated values. However, in order to keep the programming simple and the gate sizes small, it may be desirable to make a rule such as the following: if at step n the extrapolated value of $y(t)$ is used and if m past values are used in our extrapolation procedure, then at the next m steps we will use the computed value without a check. For instance, using the previous example, if at t_3 we had used y_3^E due to a malfunction then at t_4, t_5, and t_6 we would accept the computed value without an extrapolative check. Since we expect malfunctions to be rare, the above procedure is probably applicable. However, if more safety is desired, we might use some simple procedure for changing the gate size which would apply to all successive steps affected by the extrapolated value at step n.

In order to use properly the method of extrapolative checking, it is necessary to program it very carefully. As an illustration of how this may be done let us, referring to the previous example, assume that the value of ΔQ corresponding to time t_3 has been computed and is in the accumulator. Then the following procedure for the case of Lagrangian extrapolation will perform the extrapolative check and set up the conditions of the example on the use of multiple storage and computation.

1) Store the accumulator in b_1, b_2, and b_3.
2) Using the values of ΔQ corresponding to t_0, t_1, and t_2 and (2) find the extrapolated value of ΔQ and store in c_1, c_2, and c_3.
3) Now using the values of ΔQ at t_1 and t_2 and (11) extrapolate to find ΔQ and leave this value in the accumulator.
4) Compare the two extrapolated values using $C(c_3)$ for the first extrapolation. If the difference falls within the gate go on to step 5). If not assume the computed ΔQ is correct and go to step 7).
5) Compare the magnitude of the difference between the extrapolated and computed values in b_3 and c_3 with the associated gate. If the magnitude is less than the gate size use the contents b_1, b_2, and b_3 for ΔQ and go to step 7). If not, replace the contents of b_1, b_2, and b_3 by the contents of c_1, c_2, and c_3.
6) If the extrapolated rather than the computed value has been used, set up the desired modification of the extrapolative check in succeeding steps.
7) "Age" the data. That is, set up the extrapolation for the next cycle.

This sequence assures that any single machine error (and some multiple errors), that causes an error larger than the associated gate size in the computation for ΔQ or the check computation will be detected and corrected. The maximum possible difference between the final value of ΔQ in b_1, b_2, and b_3 and the correct computed value is given by (5). In the Appendix the above sequence is written out order by order. For step 6) we have used the case in which no extrapolative check is made at the three computations of $y(t)$ following the one in which an extrapolated rather than a computed value was used. The number of program steps needed when no error has been made (the usual case) is thirty-two (1–26 and 38–43). Thus if this sequence is used to check a lengthy computation it will take up only a small percentage of the total computation time when no malfunction occurs. Of course, the number of program steps required when an error has been made in the computed value must be taken into account. This total is 41 (1–35 and 38–43).

The formulas that have been included here clearly cover some of the simplest cases of the method of extrapolative checking. Use of more previous values of the function would naturally require more complicated formulas which, as we have pointed out, may be derived in a fashion similar to those listed here. Of course, the very simplicity of the formulas we have displayed is a large part of their charm, for the simpler they are the less time the check takes and the less program storage is used by the check routine.

By Scaling

When the change of a single bit can greatly increase the magnitude of a small quantity, we have a serious source of computer errors. In cases where the total variation of a quantity is over a small range the possibility of such an error can be greatly lessened by proper use of scaling. For example, again using ΔQ of the previous examples, suppose that we know that the maximum magnitude of ΔQ will never exceed some small quantity. Then if we properly scale up the quantities used in computing ΔQ so that the maximum magnitude of ΔQ would use all the positions in a word, then no single-bit change can greatly increase the magnitude of ΔQ by more than the maximum of $\Delta Q/2$. (Incidentally this scaling up may also be useful in improving the accuracy of the calculation.) For instance if the computer is such that all numbers have absolute value less than one, then we would scale ΔQ so that its maximum scaled absolute value is as near one as possible. Thus the scaling process acts as a type of limiter.

It often becomes necessary to scale down previously scaled up quantities for use in subsequent calculations. For example in order to add ΔQ to Q, ΔQ must be scaled down (barring the case where the single quantity Q would be stored as a multiple word length number). This scaling down is generally a matter of shifting. It may be desirable to check that there has been no error made in this shifting process. This is particularly true when the first k significant digits of Q are vital. For the case of a number ΔQ in register b_1 the following procedure might be used:

1) Clear and add b_1, shift right k places and store the result in b_1.
2) Clear and add b_1, shift left k places and test for overflow. If there is overflow it means that the shifting in step 1) was perhaps done improperly and we must shift again.

By the Use of Built-in Machine Checking Devices

The detection of overflow in real time computers may be used in a fashion similar to its use in nonreal time computers where typically the overflow indication sets a flip-flop which may be tested by a transfer on overflow instruction. An example of the use of overflow to detect arithmetic errors was given in the section on scaling. In general the detection of an arithmetic error by overflow would send the program to a correction routine. In the case of the scaling example the correction would merely provide for the rescaling of the quantity. Overflow detection may also be used to indicate when the magnitude of a quantity has exceeded the capacity of the computer and, therefore, must be scaled down. In this case a scaling routine would be used in conjunction with the transfer on overflow instruction.

Parity-check failures, however, which typically stop nonreal time computers must clearly be handled differently on real time computers. One possible scheme is to have a parity-check failure set a flip-flop as in the case of overflow. An instruction would then be provided to, for example, transfer if a parity failure had occurred since the last use of this instruction. The usefulness of such an instruction in real-time applications would probably be at best limited since continual testing for parity failures is too costly in time. On the other hand, occasional testing creates a problem in knowing what to do if a failure is detected, since the location and time of the occurrence cannot be determined. A second scheme for the use of parity failures would be for such a failure to cause an automatic transfer to a specified location and to cause the contents of the program counter to be put in a specified register. A correction routine starting in the transferred to location would then decide what action to take depending on the location of the failure. This scheme seems much more useful than the other for real-time applications.

In general we may say that built-in machine checking devices must, on the detection of an error, cause an action which will not stop the computer but which will make the fact of the error available to the computer in some form.

Address Coding

This is a technique in digital computer programming which has applications to both arithmetic and sequencing errors. Basically the method involves coding the addresses of the locations of a group of quantities so that no single bit change in one address will give another address in the group. In this sense the method uses a philosophy similar to that of the error detecting and correcting codes used in transmitting information [4].

The method is best illustrated by an example. Consider the case where the program in the computer is divided into a number of different computation blocks. Let us assume that the determination of which block of computation is to be performed at a given time is determined by a control routine which makes its decisions on the basis of the present state of the system. Associated with this control routine will be a number of transfer instructions to the various computation blocks. We may consider these arranged in a table. The basic function of the control program is then to calculate the correct entry to this table and then, using some type of address modification facility, to enter the table and so transfer to the desired computation block. Clearly an error in entering the table which causes a transfer to the wrong block would violate the proper sequence of the program and would cause a serious or vital error. The procedure about to be described insures that any single bit change in the address of the correct table entry (which, for instance, might be caused by a malfunction of the address modification facility) and many multiple-bit errors will be detected and then the program will enter a correction procedure. Although this method may be somewhat costly in storage, it is fast and easy to implement.

Suppose the table has eight entries. Then we will store these eight entries in eight out of sixteen consecutive registers, where the address of the first register is a multiple of 16, in such a way that a single-bit change in any one of the eight addresses will not give another address of the set. (In the general case for k entries we will use a sequence of 2^n registers, the first address of which will be a multiple of 2^n, where $2^{n-1} < 2k \leq 2^n$.) We do this by choosing the eight of the sixteen registers whose addresses have an even number of ones in the four least significant places. The number of ones in the more significant places is a constant in these sixteen registers. In the general case we would choose the 2^{n-1}

of the 2^n registers with an even number of ones in the n least significant places.

To show that the above is possible we must prove that exactly 2^{n-1} of the binary numbers from 0 to 2^{n-1} have an even number of ones. Now the number of binary numbers between 0 and 2^{n-1} having m ones is $\binom{n}{m}$ since 2^{n-1} has n significant digits. Thus the number having an even number of ones is

$$\sum_{\substack{m=0 \\ m \text{ even}}}^{n} \binom{n}{m} = \frac{1}{2}\left[\sum_{m=0}^{n}\binom{n}{m} + \sum_{m=0}^{n}(-1)^m\binom{n}{m}\right]$$
$$= \frac{1}{2}\,[2^n + 0] = 2^{n-1} \tag{13}$$

which completes the proof.

Now if we calculate the entry to the table as a number between zero and fifteen (to be used to modify the four least significant digits of the address of the first register in the table) then any single-bit error in this calculation or the subsequent address modification will give an address in the table other than one of the eight correct entries. If in the other eight registers we store a transfer instruction, the same one in each register, which when executed transfers control to a correction routine, then this single-bit error will cause the correction routine to be entered. This routine then would provide for the recalculation of the entry address. This procedure clearly will also catch various multiple-bit errors. It should be realized, however, that single-bit errors in the non-modified digits of the address of the first register in the table will not be detected by this procedure. However, this type of error can be made extremely unlikely if the instructions of the program are stored in some permanent fashion.

An apparent drawback to this procedure is the difficulty of calculating the entry address. In the usual case of eight entries stored in eight consecutive registers it is merely a matter of calculating a number between zero and seven and then using this to modify the address of the first register in the table. However, with address coding the eight entries will be in the 0, 3, 5, 6, 9, 10, 12, and 15 positions in the sixteen entry table. Thus knowing we want the jth entry where j is between 0 and 7 means calculating the relationship between j and the position of the jth entry in the address coded table. This can be simply done by the following procedure:

1) Shift the number j left one place
2) To this result logically add (*i.e.*, bit by bit addition or addition without carry) j.

The result is then a unique entry in the table. To see this we need only note that the shifted j and j itself contain the same number of ones so that the logical sum contains an even number of ones. The result is less than 2^n and thus does correspond to an entry in the table. Finally it is easy to see that two different j's cannot give the same logical sum as above. In particular this process transforms the numbers 0, 1, $\cdots$, 7 into 0, 3, 6, 5, 12, 15, 10, 9 respectively.[10] Thus the second entry in the original table is in register 3 in the address-coded table and the sixth entry in the original table is in register 15 of the address-coded table. When the number of entries in the table is not a power of two, more than half of the entries in the address-coded table must be transfers to the correction routine.

A similar application of address coding which detects and corrects arithmetic errors without being so wasteful of storage is the case of table look-up of numbers. Let us assume we have two tables of constants to store where all the entries in each table have the same sign. Then we can store the entries of one table in registers whose addresses have an even number of ones and the entries of the other table in registers whose addresses have an odd number of ones. In order to distinguish between the two tables we need only store one table with positive signs and one with negative signs. Then when an entry is extracted from the table we test the sign and if not correct we know that a single-bit (or perhaps a multiple-bit) error has occurred so we then go to a correction routine which again provides for recalculation of the address of the table entry. In this application of the method of address coding the only storage wasted is the difference between 2^n and the total of entries in both tables, where the number of entries in each table is less than or equal to 2^{n-1} but greater than 2^{n-2} for at least one table. But even some of these registers may be used for constants and thus not wasted.

Address coding has applications beyond table look-up operations. In general we can say that whenever there is a set of two or more locations in a digital computer such that the use of the number in, or the transfer to, one member of the set other than the desired member of the set would cause a serious or vital error, then the use of address coding may be profitable. One further example of this method will illustrate this general property. As in a previous example let us consider the case where the program is divided up into a number of blocks of computation. In order to increase the efficiency of block-entry checking (to be described later) it is desirable to have the entry addresses of the various blocks as different as possible, where by this we mean that each block entry address will differ in as many positions as possible from all the other block entry addresses. To achieve this greatest possible difference is clearly a type of address coding and moreover it is an application which costs nothing in time or storage.

[10] It is interesting to note that the sequence: 0, 3, $\cdots$, 9, is made up of the first, third, fifth, etc. members of a 4-bit Gray Code[5] in their proper sequence. To get the second, fourth, sixth, etc. members of the Gray Code we transform 0, 1, $\cdots$, 7, as before and then logically add a one in the least significant bit position. This transformation gives a sequence of numbers each of which has an odd number of ones.

Other Techniques of Detecting and Correcting Sequencing Errors

By the Cyclic Method of Calculation and the Use of Timing Pulses

In real-time computers the use of periodic timing pulses is of course a common method of keeping the computer synchronized with the rest of the system. This timing pulse may also be used to prevent the admittedly unlikely possibility that the program will get so badly out of sequence as to not be able to get back by itself (*e.g.*, in a loop). To use a timing pulse for this purpose we merely require that there be a timing pulse at a fixed interval which automatically transfers control to a fixed point in the program. Thus the program would be assured of being at a certain point at fixed time intervals determined by the interval, T, between timing pulses. Clearly this scheme necessitates breaking up the program into cycles, each of which takes no more than T seconds to perform. Possible excess time at the end of a cycle might be used for diagnostic checking. This cyclic method of calculation has the added advantage of simplifying prediction and extrapolation formulas by enabling us to calculate quantities at fixed intervals of time, which we assumed was the case in discussing extrapolative checking.

By Block-Entry Checking

In order to assure that a block of calculations has been correctly entered the following simple sequence of program steps may be used:

At the entry to the block:

1) Clear and add b_i—where b_i is any constant.
2) Store in register R—to set up check.

At the end of the block·

1) Clear and add $C(R)$
2) Subtract b_i
3) Store in R—in order to reset R
4) Jump on nonzero to correction program.

The constant b_i must be different for each block but, since the value of b_i is immaterial, constants already in the program may be used. This procedure is insurance that *a block* was entered at the beginning. The address-coding procedure mentioned in the previous section is insurance that the *wrong block* will not be entered at its correct entry place.

By the Use of Parity Checking

Incorrect readout of instructions from memory can easily cause sequencing errors. Parity checking is of course a standard method of detecting such errors. As to the correction of such an error, the simple approach seems to be the best one; that is, when a parity-check failure is detected on read-out, we merely read out the desired instruction (or number) again. Some limit to the number of times a given read-out will be attempted should probably be made since a continued parity-check failure would indicate a dropped bit in storage (or perhaps a failure in a read circuit).

The Relation Between Programmed Error Detection and Correction, Component Reliability and Checking Circuitry

As with all the elements in a real-time control system, the digital computer must be an inherently reliable piece of equipment in order for the system to operate usefully. By emphasizing the treatment of single malfunctions we have implicitly assumed a basically reliable computer. Indeed programmed error detection and correction can only supplement the use of reliable computer components. It cannot replace them. But this supplementing may in fact be the difference between success and failure since as we have pointed out a single malfunction may mean catastrophe. This is especially true when maintenance of the computer is difficult such as in the case of computers in military systems in the field.

The use of checking circuitry is another method of improving the performance of a digital computer. It does not however negate the need for programmed error detection and correction which is inherently more flexible and more easily applicable to special situations than checking circuitry. Furthermore in some cases, notably in military applications, the extra size and weight that checking circuitry adds to a computer may make it impractical.

It is certainly not the purpose of this paper to discuss in any detail the problem of reliable computer components. But it is perhaps worthwhile to mention at least two places in a computer where reliable components and/or checking circuitry are of special importance in that malfunctions in these components cause errors which are particularly serious or which are difficult to detect by programming techniques.

Mutilation anywhere in that part of the memory where instructions or constants of the program are stored is certainly very serious. Moreover mutilation in certain places can produce vital errors which cannot be corrected by programming techniques. To prevent this type of malfunction some type of permanent (*e.g.*, wired-in) storage is desirable. Although this may not be practical in general purpose computers, on real time computers, where typically a single program is used over long periods of time, it is eminently practical.

Even very occasional malfunctions of the program counter can cause very serious errors which may not be detected. One possible method of improving the reliability of the program counter is, analogously to the method of triple computation and storage, to provide three program counters and, when all three do not agree, to choose as correct a value that appears in two of them. This method, however, appears quite expensive to implement.

In general it can be said that a digital computer is extremely sensitive to malfunctions anywhere in the control unit of the computer for such machine errors may be undetectable or uncorrectable. This statement only serves to illustrate what we said at the beginning of this section; namely, that programmed error detection and correction can only supplement but can never replace reliable computer components.

Conclusion

A knowledge of error-detection and error-correction techniques is necessary but not sufficient to the writing of an efficient program for a real-time digital computer. We have already indicated that it is also necessary to have a thorough knowledge of the computer in question with regard to the probabilities of malfunction in the various parts of the computer. Furthermore the system of which the computer is a part must be clearly understood in order that the damage to the system operation of each type of malfunction may be assessed. It is the implication of this last factor which often determines the extent of the error detecting and correcting that is programmed. Clearly the cost in money of providing the extra storage (and perhaps also speed) in the computer for error detection and correction must be weighed against both the probability of failure of the system without error detection and correction and the cost of such failure. In this connection we must consider a given error-detection and correction technique in relation both to the probability that the error this technique is supposed to detect and correct will actually occur and to the probability that the technique will detect and correct the error if it does occur. Also we must consider factors such as whether the added weight of and space occupied by the extra storage needed for error detection and correction will impede the operation of the system. Thus, in general, the determination of what error detecting and correcting to build into a program is a matter of weighing a number of factors: economic, mathematical, and physical.

Virtually all the techniques we have presented here still must be tested in actual real-time situations. Until this has been done the effectiveness of these methods cannot be fairly assessed. However, it seems reasonable to say that some of the techniques will have useful application in a wide class of real-time systems. This should be notably true of extrapolative checking which is a powerful and simple method of checking the reasonableness of the result of a lengthy computation. Other techniques, such as address coding, will be useful only in those cases where the basic structure of the program makes them applicable. Some techniques are clearly aimed at more unlikely types of errors than others and thus will find only specialized application.

For most of the methods described here only one or two of the possible applications have been mentioned. It would seem likely that the ingenuity of individual programmers will find a number of applications for a method such as address coding. And of course the ingenuity of individual programmers is basic to good programmed error detection and correction since the specialized nature of most real-time systems makes them ripe for the use of very specialized techniques. In this paper we have tried to present some of the general techniques of error detection and correction and to indicate the path toward some of the more specialized techniques.

The use of real-time digital computers is a young and rapidly expanding field. With the development of larger and more versatile digital computers for use in larger and more complicated real time control systems, there will come a growing need for more efficient and more sophisticated programmed error-detection and error-correction techniques.

Acknowledgment

Most of the ideas presented in this paper were generated in a seminar at the Bell Telephone Laboratories. The author would like to thank the other regular members of that seminar, E. G. Andrews, H. A. Helm, N. K. Poole, J. G. Tryon, and V. M. Wolontis, for the ideas they have contributed to this paper.

Appendix

A Routine for Extrapolative Checking

The following routine performs the operations described in the seven steps at the end of the section on extrapolative checking. The following storage locations are assumed:

A) q_0, q_1, and q_2 for the values of ΔQ corresponding to t_0, t_1, and t_2.
B) g_1, g_2, $\cdots$, g_5 for respectively, the gate sizes to check the extrapolation and the computation and the constants 0, 1, and 3.
C) t_1 for temporary storage.
D) k_1 for a counter, initially containing 3.

1) Store in	b_1	
2) Store in	b_2	Step 1)—Puts ΔQ in b_1, b_2, b_3.
3) Store in	b_3	
4) Clear and add	k_1	Transfer only if the extrapolative check is not to be used (see 33–37).
5) Transfer if nonzero to	36	
6) Clear and add	q_2	
7) Subtract	q_1	
8) Multiply by	g_5	Step 2)—Computes the extrapolated value and stores it in c_1, c_2, c_3.
9) Add	q_0	
10) Store in	c_1	
11) Store in	c_2	
12) Store in	c_3	
13) Clear and add	q_2	Step 3)—Computes second extrapolated value of ΔQ.
14) Add	q_2	
15) Subtract	q_1	
16) Subtract	c_3	
17) Store in	t_1	
18) Clear and add absolute value of	t_1	Step 4)—Tests extrapolated values, if result plus test fails.
19) Subtract	g_1	
20) Transfer on plus to	38	
21) Clear and add	b_3	
22) Subtract	c_3	
23) Store in	t_1	Step 5)
24) Clear and add	g_2	a) Compares computed and extrapolated values. If result plus use computed value.
25) Subtract absolute value of	t_1	
26) Transfer on plus to	38	
27) Clear and add	c_1	
28) Store in	b_1	
29) Clear and add	c_2	b) Replaces computed values by extrapolated values.
30) Store in	b_2	
31) Clear and add	c_3	
32) Store in	b_3	
33) Clear and add	g_5	Step 6)
34) Store in	k_1	a) Disables extrapolative check by putting nonzero quantity in k_1.
35) Transfer to	38	
36) Subtract	g_4	b) Reduces counter by 1. After three cycles counter contains 0.
37) Store in	k_1	
38) Clear and add	q_1	
39) Store in	q_0	
40) Clear and add	q_2	Step 7—"Ages" data for use in next extrapolation.
41) Store in	q_1	
42) Clear and add	b_3	
43) Store in	q_2	

Notes:

1) The number of program steps executed in this routine is:
   a) 32 if the extrapolative check is used and no errors are made.
   b) 41 if the extrapolative check is used and the computed value is in error.
   c) 26 if the extrapolative check is used and one of the extrapolations is in error.
   d) 13 if the extrapolative check is disabled.
2) Another possible method of disabling the check would be to store temporarily a transfer instruction after the third order in the routine. However, in real-time computers it is very desirable to have the instructions of the program permanently stored and therefore untouchable by the program itself.
3) Steps 38)–43) must be executed whether the check is disabled or not so that when the check is reinstated the proper data will be available.
4) It is reasonably clear that performing step 6) by any method more sophisticated than disabling the check would require quite a large number of extra program steps.
5) The contents of k_1 has been made equal to 3 initially to indicate the case in which no extrapolative check is made for the first three cycles.
6) The author would like to thank J. G. Tryon for his help in streamlining this routine.

Bibliography

[1] Doersam, C. H. Jr. "The Reasonableness Check in Automation," 1956 IRE Convention Record, Part 4, pp. 67–72.
[2] Hildebrand, F. B. *Introduction to Numerical Analysis*, New York: McGraw-Hill Book Company, Inc., 1956.
[3] Kopal, Z. *Numerical Analysis*, New York: John Wiley and Sons, Inc., 1955.
[4] Hamming, R. W. "Error Detecting and Error Correcting Codes," *Bell System Technical Journal*, Vol. 29 (April, 1950), pp. 147–160.
[5] Flores, I. "Reflected Number Systems," IRE Transactions on Electronic Computers, Vol. EC-5 (June, 1956), pp. 79–82.

The FORTRAN Automatic Coding System

J. W. BACKUS†, R. J. BEEBER†, S. BEST‡, R. GOLDBERG†, L. M. HAIBT†, H. L. HERRICK†, R. A. NELSON†, D. SAYRE†, P. B. SHERIDAN†, H. STERN†, I. ZILLER†, R. A. HUGHES§, and R. NUTT||

Introduction

THE FORTRAN project was begun in the summer of 1954. Its purpose was to reduce by a large factor the task of preparing scientific problems for IBM's next large computer, the 704. If it were possible for the 704 to code problems for itself and produce as good programs as human coders (but without the errors), it was clear that large benefits could be achieved. For it was known that about two-thirds of the cost of solving most scientific and engineering problems on large computers was that of problem preparation. Furthermore, more than 90 per cent of the elapsed time for a problem was usually devoted to planning, writing, and debugging the program. In many cases the development of a general plan for solving a problem was a small job in comparison to the task of devising and coding machine procedures to carry out the plan. The goal of the FORTRAN project was to enable the programmer to specify a numerical procedure using a concise language like that of mathematics and obtain automatically from this specification an efficient 704 program to carry out the procedure. It was expected that such a system would reduce the coding and debugging task to less than one-fifth of the job it had been.

Two and one-half years and 18 man years have elapsed since the beginning of the project. The FORTRAN system is now complete. It has two components: the FORTRAN language, in which programs are written, and the translator or executive routine for the 704 which effects the translation of FORTRAN language programs into 704 programs. Descriptions of the FORTRAN language and the translator form the principal sections of this paper.

The experience of the FORTRAN group in using the system has confirmed the original expectations concerning reduction of the task of problem preparation and the efficiency of output programs. A brief case history of one job done with a system seldom gives a good measure of its usefulness, particularly when the selection is made by the authors of the system. Nevertheless, here are the facts about a rather simple but sizable job. The programmer attended a one-day course on FORTRAN and spent some more time referring to the manual. He then programmed the job in four hours, using 47 FORTRAN statements. These were compiled by the 704 in six minutes, producing about 1000 instructions. He ran the program and found the output incorrect. He studied the output (no tracing or memory dumps were used) and was able to localize his error in a FORTRAN statement he had written. He rewrote the offending statement, recompiled, and found that the resulting program was correct. He estimated that it might have taken three days to code this job by hand, plus an unknown time to debug it, and that no appreciable increase in speed of execution would have been achieved thereby.

† Internat'l Business Machines Corp., New York, N. Y.
‡ Mass. Inst. Tech., Computation Lab., Cambridge, Mass.
§ Radiation Lab., Univ. of California, Livermore, Calif.
|| United Aircraft Corp., East Hartford, Conn.

The FORTRAN Language

The FORTRAN language is most easily described by reviewing some examples.

Arithmetic Statements

Example 1: Compute:

$$\text{root} = \frac{-(B/2) + \sqrt{(B/2)^2 - AC}}{A}.$$

FORTRAN Program:

```
ROOT
  = (-(B/2.0) + SQRTF((B/2.0) * *2 - A*C))/A.
```

Notice that the desired program is a single FORTRAN statement, an arithmetic formula. Its meaning is: "Evaluate the expression on the right of the = sign and make this the value of the variable on the left." The symbol * denotes multiplication and * * denotes exponentiation (*i.e.*, A * *B means A^B). The program which is generated from this statement effects the computation in floating point arithmetic, avoids computing (B/2.0) twice and computes (B/2.0) * *2 by a multiplication rather than by an exponentiation routine. [Had (B/2.0) * *2.01 appeared instead, an exponentiation routine would necessarily be used, requiring more time than the multiplication.]

The programmer can refer to quantities in both floating point and integer form. Integer quantities are somewhat restricted in their use and serve primarily as subscripts or exponents. Integer constants are written without a decimal point. Example: 2 (integer form) vs 2.0 (floating point form). Integer variables begin with I, J, K, L, M, or N. Any meaningful arithmetic expression may appear on the right-hand side of an arithmetic statement, provided the following restriction is observed: an integer quantity can appear in a floating-point expression only as a subscript or as an exponent or as the argument of certain functions. The functions which the programmer may refer to are limited only by those available on the library tape at the time, such as SQRTF, plus those simple functions which he has defined for the given problem by means of function statements. An example will serve to describe the latter.

Function Statements

Example 2: Define a function of three variables to be used throughout a given problem, as follows:

```
ROOTF(A, B, C)
  = (-(B/2.0) + SQRTF((B/2.0) * *2 - A*C))/A.
```

Function statements must precede the rest of the program. They are composed of the desired function name (ending in F) followed by any desired arguments which appear in the arithmetic expression on the right of the = sign. The definition of a function may employ any previously defined functions. Having defined ROOTF as above, the programmer may apply it to any set of arguments in any subsequent arithmetic statements. For example, a later arithmetic statement might be

```
THETA = 1.0 + GAMMA*ROOTF(PI, 3.2*Y
                              + 14.0, 7.63).
```

DO Statements, DIMENSION Statements, and Subscripted Variables

Example 3: Set $Q_{\max}$ equal to the largest quantity $P(a_i+b_i)/P(a_i-b_i)$ for some i between 1 and 1000 where $P(x)=c_0+c_1x+c_2x^2+c_3x^3$.

FORTRAN Program:

```
1) POLYF(X)=C0+X*(C1+X*(C2+X*C3)).
2) DIMENSION A(1000), B(1000).
3) QMAX=-1.0 E20.
4) DO 5 I=1, 1000.
5) QMAX=MAXF(QMAX, POLYF(A(I)
          +B(I))/POLYF(A(I)-B(I))).
6) STOP.
```

The program above is complete except for input and output statements which will be described later. The first statement is not executed; it defines the desired polynomial (in factored form for efficient output program). Similarly, the second statement merely informs the executive routine that the vectors A and B each have 1000 elements. Statement 3 assigns a large negative initial value to QMAX, -1.0×10^{20}, using a special concise form for writing floating-point constants. Statement 4 says "DO the following sequence of statements down to and including the statement numbered *5* for successive values of *I* from *1* to *1000.*" In this case there is only one statement 5 to be repeated. It is executed 1000 times; the first time reference is made to A(1) and B(1), the second time to A(2) and B(2), etc. After the 1000th execution of statement 5, statement 6—STOP—is finally encountered. In statement 5, the function MAXF appears. MAXF may have two or more arguments and its value, by definition, is the value of its largest argument. Thus on each repetition of statement 5 the old value of QMAX is replaced by itself or by the value of POLYF(A(I)+B(I))/POLYF (A(I)-B(I)), whichever is larger. The value of QMAX after the 1000th repetition is therefore the desired maximum.

Example 4: Multiply the $n\times n$ matrix $a_{ij}(n\leq 20)$ by its transpose, obtaining the product elements on or below the main diagonal by the relation

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k}\cdot a_{j,k} \qquad (\text{for } j \leq i)$$

and the remaining elements by the relation

$$c_{j,i} = c_{i,j}.$$

FORTRAN Program:

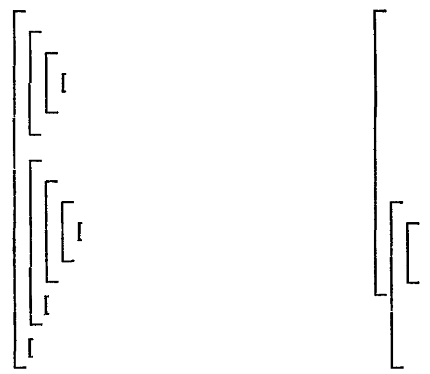
```
      DIMENSION A(20, 20), C(20, 20)
      DO 2 I = 1, N                          P
      ↓                                      ↓
      DO 2 J = 1, I                          Q
      ↓                                      ↓
      C(I, J) = 0.0
      DO 1 K = 1, N                          R
      ↓                                      ↓
1     C(I, J) = C(I, J) + A(I, K)*A(J, K)
2     C(J, I) = C(I, J)
      STOP
```

As in the preceding example, the DIMENSION statement says that there are two matrices of maximum size 20×20 named A and C. For explanatory purposes only, the three boxes around the program show the sequence of statements controlled by each DO statement. The first DO statement says that procedure P, *i.e.*, the following statements down to statement 2 (outer box) is to be carried out for I = 1 then for I = 2 and so on up to I = N. The first statement of procedure P(DO 2 J = 1, I) directs that procedure Q be done for J = 1 to J = I. And of course each execution of procedure Q involves N executions of procedure R for K = 1, 2, · · · , N.

Consider procedure Q. Each time its last statement is completed the "index" J of its controlling DO statement is increased by 1 and control goes to the first statement of Q, until finally its last statement is reached and J = I. Since this is also the last statement of P and P has not been repeated until I = N, I will be increased and control will then pass to the first statement of P. This statement (DO 2 J = 1, I) causes the repetition of Q to begin again. Finally, the last statement of Q and P (statement 2) will be reached with J = I and I = N, meaning that both Q and P have been repeated the required number of times. Control will then go to the next statement, STOP. Each time R is executed a new term is added to a product element. Each time Q is executed a new product element and its mate are obtained. Each time *P* is executed a product row (over to the diagonal) and the corresponding column (down to the diagonal) are obtained.

The last example contains a "nest" of DO statements, meaning that the sequence of statements controlled by one DO statement contains other DO statements. Another example of such a nest is shown in the next column, on the left. Nests of the type shown on the right are not permitted, since they would usually be meaningless.

Although not illustrated in the examples given, the programmer may also employ subscripted variables having three independent subscripts.



READ, PRINT, FORMAT, IF and GO TO Statements

Example 5: For each case, read from cards two vectors, ALPHA and RHO, and the number ARG. ALPHA and RHO each have 25 elements and ALPHA(I) ≤ALPHA(I+1), I = 1 to 24. Find the SUM of all the elements of ALPHA from the beginning to the last one which is less than or equal to ARG [assume ALPHA(1) ≤ ARG < ALPHA(25)]. If this last element is the *N*th, set VALUE = 3.14159 * RHO(*N*). Print a line for each case with ARG, SUM, and VALUE.

FORTRAN Program:

```
   DIMENSION ALPHA(25), RHO(25)
1) FORMAT(5F12.4).
2) READ 1, ALPHA, RHO, ARG
   SUM = 0.0
   DO 3 I = 1, 25
   IF (ARG-ALPHA(I)) 4, 3, 3.
3) SUM = SUM+ALPHA(I)
4) VALUE = 3.14159*RHO(I-1)
   PRINT 1, ARG, SUM, VALUE
   GO TO 2.
```

The FORMAT statement says that numbers are to be found (or printed) *5* per card (or line), that each number is in fixed point form, that each number occupies a field *12* columns wide and that the decimal point is located *4* digits from the right. The FORMAT statement is not executed; it is referred to by the READ and PRINT statements to describe the desired arrangement of data in the external medium.

The READ statement says "*READ* cards in the card reader which are arranged according to FORMAT statement 1 and assign the successive numbers obtained as values of *ALPHA*(I) I = 1, 25 and *RHO*(I) I = 1, 25 and *ARG*." Thus "ALPHA, RHO, ARG" is a description of a list of 51 quantities (the size of ALPHA and RHO being obtained from the DIMENSION statement). Reading of cards proceeds until these 51 quantities have been obtained, each card having five numbers, as per the FORMAT description, except the last which has the value of ARG only. Since ARG terminated the list, the remaining four fields on the last card are not read. The PRINT statement is similar to READ except that it specifies a list of only three quantities. Thus each execution of PRINT causes a single line to be printed with ARG, SUM, VALUE printed in the first three of the five fields described by FORMAT statement 1.

The IF statement says "*If ARG−ALPHA(I)* is negative go to statement 4, if it is zero go to statement 3, and if it is positive go to 3." Thus the repetition of the two statements controlled by the DO consists normally of computing ARG−ALPHA(I), finding it zero or positive, and going to statement 3 followed by the next repetition. However, when I has been increased to the extent that the first ALPHA exceeding ARG is encountered, control will pass to statement 4. Note that this statement does not belong to the sequence controlled by the DO. In such cases, the repetition specified by the DO is terminated and the value of the index (in this case I) is preserved. Thus if the first ALPHA exceeding ARG were ALPHA (20), then RHO (19) would be obtained in statement 4.

The GO TO statement, of course, passes control to statement 2, which initiates reading the 11 cards for the next case. The process will continue until there are no more cards in the reader. The above program is entirely complete. When punched in cards as shown, and compiled, the translator will produce a ready-to-run 704 program which will perform the job specified.

Other Types of FORTRAN Statements

In the above examples the following types of FORTRAN statements have been exhibited.

Arithmetic statements
Function statements
DO statements
IF statements
GO TO statements
READ statements
PRINT statements
STOP statements
DIMENSION statements
FORMAT statements.

The explanations accompanying each example have attempted to show some of the possible applications and variations of these statements. It is felt that these examples give a representative picture of the FORTRAN language; however, many of its features have had to be omitted. There are 23 other types of statements in the language, many of them completely analogous to some of those described here. They provide facilities for referring to other input-output and auxiliary storage devices (tapes, drums, and card punch), for specifying preset and computed branching of control, for detecting various conditions which may arise such as an attempt to divide by zero, and for providing various information about a program to the translator. A complete description of the language is to be found in *Programmer's Reference Manual, the FORTRAN Automatic Coding System for the IBM 704.*

Preparation of a Program for Translation

The translator accepts statements punched one per card (continuation cards may be used for very long statements). There is a separate key on the keypunching device for each character used in FORTRAN statements and each character is represented in the card by several holes in a single column of the card. Five columns are reserved for a statement number (if present) and 66 are available for the statement. Keypunching a FORTRAN program is therefore a process similar to that of typing the program.

Translation

The deck of cards obtained by keypunching may then be put in the card reader of a 704 equipped with the translator program. When the load button is pressed one gets either 1) a list of input statements which fail to conform to specifications of the FORTRAN language accompanied by remarks which indicate the type of error in each case; 2) a deck of binary cards representing the desired 704 program, 3) a binary tape of the program which can either be preserved or loaded and executed immediately after translation is complete, or 4) a tape containing the output program in symbolic form suitable for alteration and later assembly. (Some of these outputs may be unavailable at the time of publication.)

The FORTRAN Translator

General Organization of the System

The FORTRAN translator consists of six successive sections, as follows.

Section 1: Reads in and classifies statements. For arithmetic formulas, compiles the object (output) instructions. For nonarithmetic statements including input-output, does a partial compilation, and records the remaining information in tables. All instructions compiled in this section are in the COMPAIL file.

Section 2: Compiles the instructions associated with indexing, which result from DO statements and the occurrence of subscripted variables. These instructions are placed in the COMPDO file.

Section 3: Merges the COMPAIL and COMPDO files into a single file, meanwhile completing the compilation of nonarithmetic statements begun in Section 1. The object program is now complete, but assumes an object machine with a large number of index registers.

Section 4: Carries out an analysis of the flow of the object program, to be used by Section 5.

Section 5: Converts the object program to one which involves only the three index registers of the 704.

Section 6: Assembles the object program, producing a relocatable binary program ready for running. Also on demand produces the object program in SHARE symbolic language.

(*Note:* Section 3 is of internal importance only; Section 6 is a fairly conventional assembly program. These sections will be treated only briefly in what follows.)

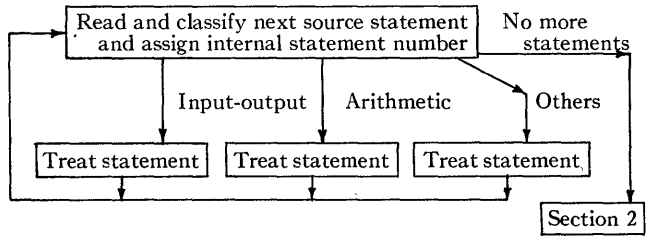
Within the translator, information is passed from section to section in two principal forms: as compiled instructions, and as tables. The compiled instructions (*e.g.*, the COMPAIL and COMPDO files, and later their merged result) exist in a four-word format which contains all the elements of a symbolic 704 instruction; *i.e.*, symbolic location, three-letter operation code, symbolic address with relative absolute part, symbolic tag, and absolute decrement. (Instructions which refer to quantities given symbolic names by the programmer have those same names in their addresses.) This symbolic format is retained until section 6. Throughout, the order of the compiled instructions is maintained by means of the symbolic locations (internal statement numbers), which are assigned in sequential fashion by section 1 as each new statement is encountered.

The tables contain all information which cannot yet be embodied in compiled instructions. For this reason the translator requires only the single scan of the source program performed in section 1.

A final observation should be made about the organization of the system. Basically, it is simple, and most of the complexities which it does possess arise from the effort to cause it to produce object programs which can compete in efficiency with hand-written programs. Some of these complexities will be found within the individual sections; but also, in the system as a whole, the sometimes complicated interplay between compiled instructions and tables is a consequence of the desire to postpone compiling until the analysis necessary to produce high object-program efficiency has been performed.

Section 1 (Beeber, Herrick, Nutt, Sheridan, and Stern)

The over-all flow of section 1 is



For an input-output statement, section 1 compiles the appropriate read or write select (RDS or WRS) instruction, and the necessary copy (CPY) instructions (for binary operations) or transfer instructions to prewritten input-output routines which perform conversion between decimal and binary and govern format (for decimal operations). When the list of the input-output statement is repetitive, table entries are made which will cause section 2 to generate the indexing instructions necessary to make the appropriate loops.

The treatment of statements which are neither input-output nor arithmetic is similar; *i.e.*, those instructions which can be compiled are compiled, and the remaining information is extracted and placed in one or more of the appropriate tables.

In contrast, arithmetic formulas are completely treated in section 1, except for open (built-in) subroutines, which are added in section 3; a complete set of compiled instructions is produced in the COMPAIL file. This compilation involves two principal tasks: 1) the generation of an appropriate sequence of arithmetic instructions to carry out the computation specified by the formula, and 2) the generation of (symbolic) tags for those arithmetic instructions which refer to subscripted variables (variables which denote arrays) which in combination with the indexing instructions to be compiled in section 2 will refer correctly to the individual members of those arrays. Both these tasks are accomplished in the course of a single scan of the formula.

Task 2) can be quickly disposed of. When a subscripted variable is encountered in the scan, its subscript(s) are examined to determine the symbols used in the subscripts, their multiplicative coefficients, and the dimensions of the array. These items of information are placed in tables where they will be available to section 2; also from them is generated a subscript combination name which is used as the symbolic tag of those instructions which refer to the subscripted variable.

The difficulty in carrying out task 1) is one of *level;* there is implicit in every arithmetic formula an order of computation, which arises from the control over ordering assigned by convention to the various symbols (parentheses, $+$, $-$, $*$, $/$, etc.) which can appear, and this implicit ordering must be made explicit before compilation of the instructions can be done. This explicitness is achieved, during the formula scan, by associating with each operation required by the formula a *level number*, such that if the operations are carried out in the order of increasing level number the correct sequence of arithmetic instructions will be obtained. The sequence of level numbers is obtained by means of a set of rules, which specify for each possible pair formed of operation type and symbol type the increment to be added to or subtracted from the level number of the preceding pair.

In fact, the compilation is not carried out with the raw set of level numbers produced during the scan. After the scan, but before the compilation, the levels are examined for empty sections which can be deleted, for permutations of operations on the same level which will reduce the number of accesses to memory, and for redundant computation (arising from the existence of common subexpressions) which can be eliminated.

An example will serve to show (somewhat inaccurately) some of the principles employed in the level-analysis process. Consider the following arithmetic expression:

$$A + B**C*(E + F).$$

In the level analysis of this expression parentheses are in effect inserted which define the proper order in which the operations are to be performed. If only three implied levels are recognized (corresponding to +, * and **) the expression obtains the following:

$$+(*(**A))+(*(**B**C)*[+(*(**E))+(*(**F))]).$$

The brackets represent the parentheses appearing in the original expression. (The level-analysis routine actually recognizes an additional level corresponding to functions.) Given the above expression the level-analysis routine proceeds to define a sequence of new dependent variables the first of which represents the value of the entire expression. Each new variable is generated whenever a left parenthesis is encountered and its definition is entered on another line. In the single scan of the expression it is often necessary to begin the definition of one new variable before the definition of another has been completed. The subscripts of the u's in the following sets of definitions indicate the order in which they were defined.

$$u_0 = + u_1 + u_3$$
$$u_1 = * u_2$$
$$u_2 = ** A$$
$$u_3 = * u_4 * u_5$$
$$u_4 = ** B ** C$$
$$u_5 = + u_6 + u_8$$
$$u_6 = * u_7$$
$$u_7 = ** E$$
$$u_8 = * u_9$$
$$u_9 = ** F.$$

This is the point reached at the end of the formula scan. What follows illustrates the further processing applied to the set of levels. Notice that u_9, for example, is defined as $**F$. Since there are not two or more operands to be combined the ** serves only as a level indication and no further purpose is served by having defined u_9. The procedure therefore substitutes F for u_9 wherever u_9 appears and the line $u_9 = **F$ is deleted. Similarly, F is then substituted for u_8 and $u_8 = *F$ is deleted. This elimination of "redundant" u's is carried to completion and results in the following:

$$u_0 = + A + u_3$$
$$u_3 = * u_4 * u_5$$
$$u_4 = ** B ** C$$
$$u_5 = + E + F.$$

These definitions, read up, describe a legitimate procedure for obtaining the value of the original expression. The number of u's remaining at this point (in this case four) determines the number of intermediate quantities which *may* need to be stored. However, further examination of this case reveals that the result of u_3 is in the accumulator, ready for u_0; therefore the store and load instructions which would usually be compiled between u_3 and u_0 are omitted.

Section 2 (Nelson and Ziller)

Throughout the object program will appear instructions which refer to subscripted variables. Each of these instructions will (until section 5) be tagged with a symbolic index register corresponding to the particular subscript combination of the subscripts of the variable [*e.g.*, (I, K, J) and (K, I, J) are two different subscript combinations]. If the object program is to work correctly, every symbolic index register must be so governed that it will have the appropriate contents at every instant that it is being used. It is the source program, of course, which determines what these appropriate contents must be, primarily through its DO statements, but also through arithmetic formulas (*e.g.* $I = N+1$) which may define the values of variables appearing in subscripts, or input formulas which may read such values in at object time. Moreover, in the case of DO statements, which are designed to produce loops in the object program, it is necessary to provide tests for loop exit. It is these two tasks, the governing of symbolic index registers and the testing of their contents, which section 2 must carry out.

Much of the complexity of what follows arises from the wish to carry out these tasks optimally; *i.e.*, when a variable upon which many subscript combinations depend undergoes a change, to alter only those index registers which really require changing in the light of the problem flow, and to handle exits correctly with a minimum number of tests.

If the following subscripted variable appears in a FORTRAN program

$$A(2*I + 1, 4*J + 3, 6*K + 5),$$

the index quantity which must be in its symbolic index register when this reference to A is made is

$$(c_1 i - 1) + (c_2 j - 1)d_i + (c_3 k - 1)d_i d_j + 1,$$

where c_1, c_2, and c_3 in this case have the values 2, 4, and 6; i, j, and k are the values of I, J, and K at the moment, and d_i and d_j are the I and J dimensions of A. The effect of the addends 1, 3, and 5 is incorporated in the address of the instruction which makes the reference.

In general, the index quantity associated with a subscript combination as given above, once formed, is not recomputed. Rather, every time one of the variables in a subscript combination is incremented under control of a DO, the corresponding quantity is incremented by the appropriate amount. In the example given, if K

is increased by n (under control of a DO), the index quantity is increased by $c_3 d_i d_j n$, giving the correct new value. The following paragraphs discuss in further detail the ways in which index quantities are computed and modified.

Choosing the Indexing Instructions; Case of Subscripts Controlled by DO's

We distinguish between two classes of subscript; those which are in the range of a DO having that subscript as its index symbol, and those subscripts which are not controlled by DO's.

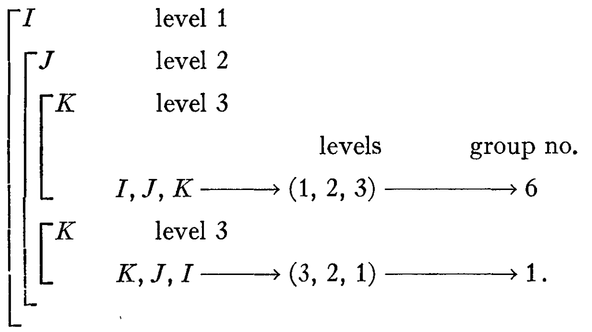
The fundamental idea for subscripts controlled by DO's is that a sequence of indexing instruction groups can be selected to answer the requirements, and that the choice of a particular instruction group depends mainly on the arrangement of the subscripts within the subscript combination and the order of the DO's controlling each subscript.

DO's often exist in *nests*. A nest of DO's consists of all the DO's contained by some one DO which is itself not contained by any other. Within a nest, DO's are assigned level numbers. Wherever the index symbol of a DO appears as a subscript within the range of that DO, the level number of the DO is assigned to the subscript. The relative values of the level numbers in a subscript combination produce a group number which, along with other information, determines which indexing instruction group is to be compiled.

The source language,

```
   DO 10 I = 1, 5
   DO 10 J = 1, 5
   DO  5 K = 1, J
5  · · · A(I, J, K) · · · (some statement referring to
       A(I, J, K))
   DO 10 K = J, 5
10 · · · A(K, J, I) · · ·
```

produces the following DO structure and group combinations:



Producing the Decrement Parts of Indexing Instructions

The part of the 704 instruction used to change or test the contents of an index register is called the decrement part of the instruction.

The decrement parts of the FORTRAN indexing instructions are functions of the dimensions of arrays and of the parameters of DO's; that is, of the initial value n_1, the upper bound n_2, and the increment n_3 appearing in the statement DO 1 $i = n_1, n_2, n_3$. The general form of the function is $[(n_2 - n_1 + n_3)/n_3] n_3 g$ where g represents necessary coefficients and dimensions, and $[x]$ denotes the integral part of x.

If all the parameters are constants, the decrement parts are computed during the execution of the FORTRAN executive program. If the parameters are variable symbols, then instructions are compiled in the object program to compute the proper decrement values. For object program efficiency, it is desirable to associate these computing instructions with the outermost DO of a nest, where possible, and not with the inner loops, even though these inner DO's may have variable parameters. Such a variable parameter (*e.g.*, N in "DO 7 $I=1$, N") may be assigned values by the programmer by any of a number of methods; it may be a value brought in by a READ statement, it may be calculated by an *arithmetic* statement, it may take its value from a *transfer* exit from some other DO whose index symbol is the pertinent variable symbol, or it may be under the control of a DO in the nest. A search is made to determine the smallest level number in the nest within which the variable parameter is not assigned a new value. This level number determines the place at which computing instructions can best be compiled.

Case of Subscripts not Controlled by DO's

The second of the two classes of subscript symbols is that of subscript symbols which are not under control of DO's. Such a subscript can be given a value in a number of ways similar to the defining of DO parameters: a value may be read in by a READ statement, it may be calculated by an arithmetic statement, or it may be defined by an exit made from a DO with that index symbol.

For subscript combinations with no subscript under the control of a DO, the basic technique used to introduce the proper values into a symbolic index register is that of determining where such definitions occur, and, at the point of definition, using a subroutine to compute the new index quantity. These subroutines are generated at executive time, if it is determined that they are necessary.

If the index quantity exists in a DO nest at the time of a transfer exit, then no subroutine calculations are necessary since the exit values are precisely the desired values.

Mixed Cases

In cases in which some subscripts in a subscript combination are controlled by DO's, and some are not, instructions are compiled to compute the initial value of the subscript combination at the beginning of the outside loop. If the non-DO-controlled subscript symbol is then defined inside the loop (that is, after the computing of the load quantity) the procedure of using a subroutine at the point of subscript definition will bring the new value into the index register.

An exception to the use of a subroutine is made when the subscript is defined by a transfer exit from a DO, and that DO is within the range of a DO controlling some other subscript in the subscript combination. In such instances, if the index quantity is used in the inner DO, no calculation is necessary; the exit values are used. If the index quantity is not used, instructions are compiled to simulate this use, so that in either case the transfer exit leaves the correct function value in the index register.

Modification and Optimization

Initializing and computing instructions corresponding to a given DO are placed in the object program at a point corresponding to the lowest possible (outermost) DO level rather than at the point corresponding to the given DO. This technique results in the desired removal of certain instructions from the most frequent innermost loops of the object program. However, it necessitates the consideration of some complex questions when the flow within a nest of DO's is complicated by the occurrence of transfer escapes from DO-type repetition and by other IF and GO TO flow paths. Consider a simple example, a nest having a DO on I containing a DO on J, where the subscript combination (I, J) appears only in the inner loop. If the object program corresponded precisely to the FORTRAN language program, there would be instructions at the entrance point of the inner loop to set the value of J in (I, J) to the initial value specified by the inner DO. Usually, however, it is more efficient to reset the value of J in (I, J) at the end of the inner loop upon leaving it, and the object program is so constructed. In this case it becomes necessary to compile instructions which follow every *transfer* exit from the inner loop into the outer loop (if there are any such exits) which will also reset the value of J in (I, J) to the initial value it should have at the entrance of the inner loop. These instructions, plus the initialization of both I and J in (I, J) at the entrance of the outer loop (on I), insure that J always has its proper initial value at the entrance of the inner loop even though no instructions appear at that point which change J. The situation becomes considerably more complicated if the subscript combination (I, J) also appears in the outer loop. In this case two independent index quantities are created, one corresponding to (I, J) in the inner loop, the other to (I, J) in the outer loop.

Optimizing features play an important role in the modification of the procedures and techniques outlined above. It may be the case that the DO structure and subscript combinations of a nest describe the scanning of a two- or three-dimensional array which is the equivalent of a sequential scan of a vector; *i.e.*, a reference to each of a set of memory locations in descending order. Such an equivalent procedure is discovered, and where the flow of a nest permits, is used in place of more complicated indexing. This substitution is not of an empirical nature, but is instead the logical result of a generalized analysis.

Other optimizing techniques concern, for example, the computing instructions compiled to evaluate the functions (governing index values and decrements) mentioned previously. When some of the parameters are constant, the functions are reduced at executive time, and a frequent result is the compilation of only one instruction, a reference to a variable, to obtain a proper initializing value.

In choosing the symbolic index register in which to test the value of a subscript for exit purposes, those index registers are avoided which would require the compilation of instructions to modify the test instruction decrement.

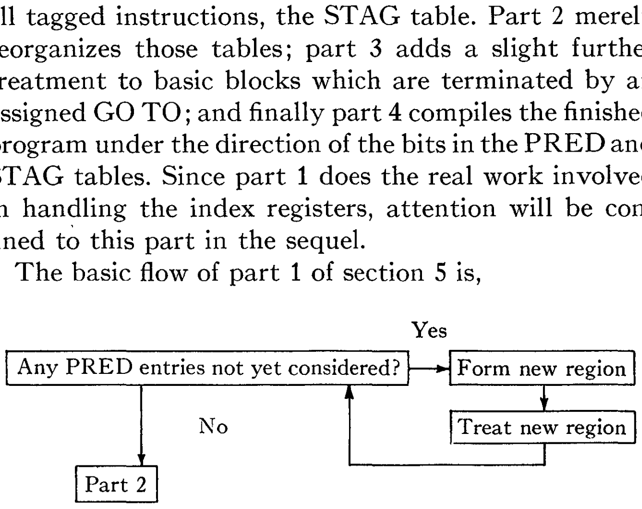
Section 4 (Haibt) and Section 5 (Best)

The result of section 3 is a complete program, but one in which tagged instructions are tagged only symbolically, and which assumes that there will be a real index register available for every symbolic one. It is the task of sections 4 and 5 to convert this program to one involving only the three real index registers of the 704. Generally, this requires the setting up, for each symbolic index register, of a storage cell which will act as an *index cell*, and the addition of instructions to load the real index registers from, and store them into, the index cells. This is done in section 5 (tag analysis) on the basis of information about the pattern and frequency of flow provided by section 4 (flow analysis) in such a way that the time spent in loading and storing index registers will be nearly minimum.

The fundamental unit of program is the *basic block;* a basic block is a stretch of program which has a single entry point and a single exit point. The purpose of section 4 is to prepare for section 5 a table of predecessors (PRED table) which enumerates the basic blocks and lists for every basic block each of the basic blocks which can be its immediate predecessor in flow, together with the absolute frequency of each such basic block link. This table is obtained by an actual "execution" of the program in Monte-Carlo fashion, in which the outcome of conditional transfers arising out of IF-type statements and computed GO TO's is determined by a random number generator suitably weighted according to whatever FREQUENCY statements have been provided.

Section 5 is divided into four parts, of which part 1 is the most important. It makes all the major decisions concerning the handling of index registers, but records them simply as bits in the PRED table and a table of all tagged instructions, the STAG table. Part 2 merely reorganizes those tables; part 3 adds a slight further treatment to basic blocks which are terminated by an assigned GO TO; and finally part 4 compiles the finished program under the direction of the bits in the PRED and STAG tables. Since part 1 does the real work involved in handling the index registers, attention will be confined to this part in the sequel.

The basic flow of part 1 of section 5 is,



Consider a moment partway through the execution of part 1, when a new region has just been treated. The less frequent basic blocks have not yet been encountered; each basic block that has been treated is a member of some region. The existing regions are of two types: transparent, in which there is at least one real index register which has not been used in any of the member basic blocks, and opaque. Bits have been entered in the STAG table, calling where necessary for an LXD (load index register from index cell) instruction preceding, or an SXD (store index register in index cell) instruction following, the tagged instructions of the basic blocks that have been treated. For each basic block that has been treated is recorded the required contents of each of the three real index registers for entrance into the block, and the contents upon exit. In the PRED table, entries that have been considered may contain bits calling for interblock LXD's and SXD's, when the exit and entrance conditions across the link do not match.

Now the PRED table is scanned for the highest-frequency link not yet considered. The new region is formed by working both forward over successors and backward over predecessors from this point, always choosing the most frequent remaining path of control. The marking out of a new region is terminated by encountering 1) a basic block which belongs to an opaque region, 2) a basic block which has no remaining links into it (when working backward) or from it (when working forward), or which belongs to a transparent region with no such links remaining, or 3) a basic block which closes a loop. Thus the new region generally includes both basic blocks not hitherto encountered, and entire regions of basic blocks which have already been treated.

The treatment of hitherto untreated basic blocks in the new region is carried out by simulating the action of the program. Three cells are set aside to represent the object machine index registers. As each new tagged instruction is encountered these cells are examined to see if one of them contains the required tag; if not, the program is searched ahead to determine which of the three index registers is the least undesirable to replace, and a bit is entered in the STAG table calling for an LXD instruction to that index register. When the simulation of a new basic block is finished, the entrance and exit conditions are recorded, and the next item in the new region is considered. If it is a new basic block, the simulation continues; if it is a region, the index register assignment throughout the region is examined to see if a permutation of the index registers would not make it match better, and any remaining mismatch is taken care of by entries in PRED calling for interblock LXD's.

A final concept is that of index register activity. When a symbolic index register is initialized, or when its contents are altered by an indexing instruction, the value of the corresponding index cell falls out of date, and a subsequent LXD will be incorrect without an intervening SXD. This problem is handled by activity bits, which indicate when the index cell is out of date; when an LXD is required the activity bit is interrogated, and if it is on an SXD is called for immediately after the initializing or indexing instruction responsible for the activity, or in the interblock link from the region containing that instruction, depending upon whether the basic block containing that instruction was a new basic block or one in a region already treated.

When the new region has been treated, all of the old regions which belonged to it simply lose their identity; their basic blocks and the hitherto untreated basic blocks become the basic blocks of the new region. Thus at the end of part 1 there is but one single region, and it is the entire program. The high-frequency parts of the program were treated early; the entrance and exit conditions and indeed the whole handling of the index registers reflect primarily the efficiency needs of these high-frequency paths. The loading and unloading of the index registers is therefore as much as possible placed in the low-frequency paths, and the object program time consumed in these operations is thus brought near to a minimum.

Conclusion

The preceding sections of this paper have described the language and the translator program of the FORTRAN system. Following are some comments on the system and its application.

Scope of Applicability

The language of the system is intended to be capable of expressing virtually any numerical procedure. Some problems programmed in FORTRAN language to date include: reactor shielding, matrix inversion, numerical integration, tray-to-tray distillation, microwave propagation, radome design, numerical weather prediction, plotting and root location of a quartic, a procedure for playing the game "nim," helicopter design, and a number of others. The sizes of these first programs range from about 10 FORTRAN statements to well over 1000, or in terms of machine instructions, from about 100 to 7500.

Conciseness and Convenience

The statement of a program in FORTRAN language rather than in machine language or assembly program language is intended to result in a considerable reduction in the amount of thinking, bookkeeping, writing, and time required. In the problems mentioned in the preceding paragraph, the ratio of the number of output machine instructions to the number of input FORTRAN statements for each problem varied between about 4 and 20. (The number of machine instructions does not include any library subroutines and thus represents approximately the number which would need to be hand coded, since FORTRAN does not normally produce programs appreciably longer than corresponding hand-coded ones.) The ratio tends to be high, of course, for problems with many long arithmetic expressions or with complex loop structure and subscript manipulation. The ratio is a rough measure of the conciseness of the language.

The convenience of using FORTRAN language is necessarily more difficult to measure than its conciseness. However the ratio of coding times, assembly program language vs FORTRAN language, gives some indication of the reduction in thinking and bookkeeping as well as in writing. This time reduction ratio appears to range also from about 4 to 20 although it is difficult to estimate accurately. The largest ratios are usually obtained by those problems with complex loops and subscript manipulation as a result of the planning of indexing and bookkeeping procedures by the translator rather than by the programmer.

Education

It is considerably easier to teach people untrained in the use of computers how to write programs in FORTRAN language than it is to teach them machine language. A FORTRAN manual specifically designed as a teaching tool will be available soon. Despite the unavailability of this manual, a number of successful courses for nonprogrammers, ranging from one to three days, have been completed using only the present reference manual.

Debugging

The structure of FORTRAN statements is such that the translator can detect and indicate many errors which may occur in a FORTRAN-language program. Furthermore, the nature of the language makes it possible to write programs with far fewer errors than are to be expected in machine-language programs.

Of course, it is only necessary to obtain a correct FORTRAN-language program for a problem, therefore all debugging efforts are directed toward this end. Any errors in the translator program or any machine malfunction during the process of translation will be detected and corrected by procedures distinct from the process of debugging a particular FORTRAN program.

In order to produce a program with built-in debugging facilities, it is a simple matter for the programmer to write various PRINT statements, which cause "snapshots" of pertinent information to be taken at appropriate points in his procedure, and insert these in the deck of cards comprising his original FORTRAN program. After compiling this program, running the resulting machine program, and comparing the resulting snapshots with hand-calculated or known values, the programmer can localize the specific area in his FORTRAN program which is causing the difficulty. After making the appropriate corrections in the FORTRAN program he may remove the snapshot cards and recompile the final program or leave them in and recompile if the program is not yet fully checked.

Experience in debugging FORTRAN programs to date has been somewhat clouded by the simultaneous process of debugging the translator program. However, it becomes clear that most errors in FORTRAN programs are detected in the process of translation. So far, those programs having errors undetected by the translator have been corrected with ease by examining the FORTRAN program and the data output of the machine program.

Method of Translation

In general the translation of a FORTRAN program to a machine-language program is characterized by the fact that each piece of the output program has been constructed, instruction by instruction, so as not only to produce an efficient piece locally but also to fit efficiently into its context as a result of many considerations of the structure of its neighboring pieces and of the entire program. With the exception of subroutines (corresponding to various functions and input-output statements appearing in the FORTRAN program), the output program does not contain long precoded instruction sequences with parameters inserted during translation. Such instruction sequences must be designed to do a variety of related tasks and are often not efficient in particular cases to which they are applied. FORTRAN-written programs seldom contain sequences of even three instructions whose operation parts alone could be considered a precoded "skeleton."

There are a number of interesting observations concerning FORTRAN-written programs which may throw some light on the nature of the translation process. Many object programs, for example, contain a large number of instructions which are not attributable to any particular statement in the original FORTRAN program. Even transfers of control will appear which do not correspond to any control statement (*e.g.*, DO, IF, GO TO) in the original program. The instructions arising from an arithmetic expression are optimally arranged, often in a surprisingly different sequence than the expression would lead one to expect. Depending on its context, the same DO statement may give rise to no instructions or to several complicated groups of instructions located at different points in the program.

While it is felt that the ability of the system to translate algebraic expressions provides an important and necessary convenience, its ability to treat subscripted variables, DO statements, and the various input-output and FORMAT statements often provides even more significant conveniences.

In any case, the major part of the translator program is devoted to handling these last mentioned facilities rather than to translating arithmetic expressions. (The near-optimal treatment of arithmetic expressions is simply not as complex a task as a similar treatment of "housekeeping" operations.) A list of the approximate number of instructions in each of the six sections of the translator will give a crude picture of the effort expended in each area. (Recall that Section 1 completely treats arithmetic statements in addition to performing a number of other tasks.)

Section Number	*Number of Instructions*
1	5500
2	6000
3	2500
4	3000
5	5000
6	2000

The generality and complexity of some of the techniques employed to achieve efficient output programs may often be superfluous in many common applications. However the use of such techniques should enable the FORTRAN system to produce efficient programs for important problems which involve complex and unusual procedures. In any case the intellectual satisfaction of having formulated and solved some difficult problems of translation and the knowledge and experience acquired in the process are themselves almost a sufficient reward for the long effort expended on the FORTRAN project.

The Interpretation and Attainment of Reliability in Industrial Data Systems

BRUCE K. SMITH†

Purpose of an Industrial Data System

THE FIRST self-sequenced digital computer was a relay device conceived for the purpose of relieving man of the tedium of involved computation. It was a docile slave, content (for the most part) to work a 24-hour day doing arithmetic faster, cheaper, and with fewer errors, than had theretofore been possible. Even before the incorporation of electronics, these capabilities were being utilized on problems which were entirely unreasonable for manual solution. The modern digital computer represents a considerably faster, more versatile, and more reliable instrument than its predecessor, but the advances in capability have been hard pressed to keep pace with the advances in application. It may truly be said at this writing that there is no foreseeable limit to either the speed or the reliability which can be profitably utilized.

Industrial data-handling systems have been conceived for the purpose of relieving man of the tedium of data gathering, data interpretation, and the use of interpreted data in factory control. We want, in essence, another docile slave who knows nothing of time clocks, and who will do our present tasks with greater accuracy, speed, economy, and reliability than presently possible through manual methods. We must expect that the introduction of such devices will permit manufacturing techniques heretofore considered unreasonable because of human limitations. We must, therefore, expect that considerably greater demands will be made of the equipment than originally intended. There is no such thing as enough reliability, or enough capability in a data-handling system which is to be modern both on the drawing board and on the day of installation. This does not imply any futility to construction with the means presently available, but rather the realization that these means can never be sufficient.

† Beckman Instruments, Inc. Fullerton, Calif.

The System as a Digital Computer

Automatic process control requires the continuing development of suitable end instruments for measurement of physical and chemical process variables. The instrumentation must, in many cases, be extremely accurate, and must in all cases produce outputs which may be scanned and monitored at a centralized facility. Complete processing of this information for a closed-loop control system necessarily implies some computing ability. Many of the simpler control tasks are being handled entirely by analog techniques. But the optimum process-control system preferably includes a digital computer, and employs the end instruments as specialized input-output equipments on the computer. It is much more profitable to consider present requirements as simplifications of that end, than to direct future growth from the somewhat hazy needs of the present.

The analogy of data-handling systems and computers is useful not only to the determination of future system requirements, but also as a means for improving present system reliability. Reliability is fundamentally more readily attainable in the digital domain, because of tolerance permission on components, and because of a higher degree of standardization in the use of those components.

Failure of a component is very often a matter of definition. We are apt to say that a unit has failed when what is meant is that it has drifted out of the tolerance range permissible in its application. When presence of energy, rather than value of energy, constitutes information, it is very much easier to design with adequate allowances for long-term variations. Early signal quantization, *i.e.*, analog-to-digital conversion, is an important step toward reliability, inasmuch as it extends tolerance permission to the majority of the system. Digital end instruments are the logical extension of this aim, but at this writing do not exist with accuracies sufficient for most system requirements.

Fundamental Component Reliability

The over-all reliability of a system is controlled by the weakest system element. Intelligent design implies not only the proper choice of every component, but a use balance which will result in all parts of the system having equivalent life expectancy. No component is unreliable except as made so by the way that it is used or by the environment in which it is employed. The most meaningful basis for comparison of components, therefore, is their worth to a proposed system when operated in the manner necessary to achieve reliability. The "manner necessary" may be one in which the device has no value whatsoever, or it may simply be one involving a maximum skin temperature or other environment-influenced factor. A number of unusual environments may be impractical to supply in a single system, but it is very often reasonable to incorporate one special environment when it can be shown to benefit most circuit elements.

The life of an electromechanical device is usually given in terms of the number of cycles of operation. Life, measured in time, is thus a direct function of use frequency. The life of an electronic component is usually given in thousands of hours. Provided dissipation is held constant, there is usually no correlation between life and the use frequency. A system employing electromechanical components may actually be more reliable (in terms of time) than one employing electronics in their stead, if the use frequency is low and if the electronic components available for substitution have a relatively short life expectancy.

Logical Balance

But straight forward comparison of electromechanical and electronic means within a given system logic is seldom an accurate basis for comparison. A properly optimized logic is one which takes recognition of component capabilities and attempts to minimize numbers of components through maximum use of these capabilities. Early digital computers comprised combinations of special-purpose logical elements, such as "multipliers," "square rooters," and various different number and command storage devices. A major step in computer reliability was taken through the use of a single arithmetic organ to perform all arithmetic functions, and the assignment of similar multiple-use tasks to other parts of the system. This required an increase in information rate to maintain a given computational speed, but it has already been noted that the speed of an electronic component need have no bearing on its life expectancy. Logical multiplexing, when the components being multiplexed are electronic, can significantly improve system reliability by virtue of reduction in the number of components required. Maximum potential reliability exists in that system which makes the greatest possible use of each of its electronic components, and the least possible use of each of its electromechanical devices.

Potential system reliability is a function both of over-all logical balance and of the detailed breakdown of that logic. A finite number of engineering man hours exists for the creation of each new system design. To a very large extent the reliability of the end product is determined by how those hours must be spread over the different design problems in the product. If the logic of a system can be implemented through combinations of a few simple common denominators, correspondingly more time is available for each denominator's development.

Modular Design

Modular design is the development and use of such basic common denominators. Because the modules are simple, and have small variety, there are also a relatively small number of different component types. This has important implications in the practicability of exhaustive component analysis, to the end of determining how they must be used to make them most reliable. It also implies that there is a greater probability of finding one optimum environment for the entire system.

Reliable module design requires not only the knowledge of how each component must be operated to assure reliability, but the prediction and control of all possible component interactions over the life of the equipment. The most reliable of basic designs may be rapidly converted into an unreliable assemblage of hardware as a result of unusual voltages or temperatures existing during breakdown of one part of the system. Prediction of all such interactions is not within human capabilities in a system which has been designed over a "practical" period of time and which is comprised of many nonstandard circuit forms. Modular design is far more than a design convenience. It is a design necessity if all factors influencing long-term life expectancy are to be considered.

Packaging

The physical manifestation of modular design is packaging, wherein each package has a logical value determined by the common denominator it contains. Modular design increases the reliability potential of a given system design. Use of a few package types to implement the logic of many different systems, varying both in speed and complexity, further increases that potential. It makes very good sense, therefore, to fix the speed and use parameters of the packages with the most ambitious system in mind. The economics of standardization are such that fast but standard packages in all applications are often cheaper than a variety of package types tailored to individual system requirements.

Packaging, and in particular packaging via printed circuit panels, provides a means for rigidly controlling the physical juxtaposition of components during manufacture. Intelligence in package design is no less important to reliability than intelligence in component selection, and is very often the deciding factor in that selection. Too often the layout of the circuit elements is delegated to a draftsman who has no comprehension of the physical or the electronic problems associated with the use of those elements. Modular design, with a few package types, makes it possible for the circuit designer to control every single factor which will influence the long-term performance of his circuit. The fact that the transformer prevented proper air flow to the resistor, so that it in turn radiated too much heat to the diode, is nobody's fault but his own. There are no design jobs in the development of a reliable package which should not receive careful engineering attention.

The Socket Is a Component

The contacts of the packages and their sockets are in the general category of components with other electromechanical devices. Like such devices, they may be used in such a way that they are extremely reliable or they may be the determining factor in machine stability. Rhodium plating has assumed the same password to reliability position in contacts as "50 per cent derating" in electronic components. Neither password will open the door to reliability unless applied as the result of a complete physical analysis of the use conditions of the component. The limiting factor in socket contact life is more often improper spring pressure than poor contact surface. For example, uniform insertion, guaranteed by close tolerance package guides, is one method of insuring constancy of design spring pressure.

Contacts, as components, should be minimized for maximum life expectancy. However, the optimum numper of contacts may very well be determined by the machine service requirements and be much more than the logical minimum. Inasmuch as service is one very important aspect of reliability, minimum number consistent with serviceability is a better statement of design objective.

The Real Meaning of Reliability and Breakdown

Reliability to a missile designer is his guarantee that nothing will fail during the first (and only) useful moments of the equipment's life. Reliability, to the general purpose computer designer, is the ratio of useful computing time to scheduled computing time. Reliability, as applied to an industrial data-control system, is much more concerned with duration of time between breakdowns. It is not sufficient to have a high probability of successful operation for some designated number of continuous hours, if the failure rate thereafter will make the equipment unusable.

Failure of a centralized process-control facility is tantamount to a complete walk out of personnel employed in manual control applications. In the analogous situation, however, it is very often possible to divert other employees to the control function. With an automatic control it is probable that the manual control facilities will be nonexistent—particularly if the automatic control capabilities have been utilized in manufacturing processes not possible with manual techniques. Insofar as most processes have sufficient inertia to go on for short times without control, breakdown is serious mainly if it is of longer duration than system inertia will carry, or if it produces a drastic change in a controlled device at the instant of breakdown.

Breakdown, then, may have two meanings. If the janitor accidentally shuts down system power, but immediately restores it, and if no information vital to control is lost during the momentary shutdown, the breakdown may be completely inconsequential to the controlled process. A system designed so that failures may be instantly remedied by a passing janitor is, perhaps, an unrealizable goal. The degree of realization, however, is one measure of effective system reliability.

Finding the Trouble

Localization of a faulty package is very much easier than location of a faulty component in an unpackaged design. If relatively few package types are involved, a small number of spares may be economically considered as a part of the system, to be used as replacements for those causing breakdown. Subsequent location of the specific component causing failure may be done on independent package checkers, involving no system down time whatever. This aspect of servicing, however, is properly the province of the original manufacturer.

Continued package reliability may be guaranteed only by assuring that component replacement is made to original circuit specification. This applies both to the choice of component and to the way that it is mounted. A package design which requires that components must be unsoldered for testing imposes unnecessarily severe conditions on those parts and the package might better be discarded than "salvaged" as a future source of difficulty.

Self Diagnosis

Even with packaging and with elaborate test indicators, fault localization in a large system demands intelligence and time from a serviceman, and assumes his availability at the time of breakdown. The expansion of a basic scanning and monitoring facility to one providing complete factory control would appear to increase this problem in direct proportion to system elaboration. Fortunately, the inclusion of a digital computer as the heart of the data-handling facility builds in the means for intelligent and rapid failure diagnosis. The concept of self diagnosis, applied to packaged digital computers, has only recently been investigated by the manufacturers of such equipments. It has been demonstrated that it is possible for a computer to completely analyze the vast majority of its own breakdowns as well as those in tertiary equipments, and to print out the results of the analysis, as the bad package number, on a directly connected typewriter. Complete introspective ability may require that the equipment involved in typeout be independent of the main control line. But extra hardware of this nature is a small price to pay for fast return to service.

These concepts are being applied to the design of business computers under development today. They constitute computing routines, rather than hardware, and as such represent an actual manufacturing saving over the conventional servicing provisions. So far, the application of such methods has been extended only to service of unchecked machines. In a business application where initial machine cost is an important factor, and where breakdown constitutes inconvenience rather than catastrophe, such an approach is a tremendous but sufficient advance over older servicing methods. As applied to factory control, however, it must be recognized that the cost of breakdown may exceed the worth of the control system, and that elaboration which materially improves reliability is cheap insurance.

Error Detection

It is profitable, therefore, to consider the logical extension of this automatic servicing procedure. The techniques for self-checking are well known to the art, having been applied in computers for over a decade and a half. Breakdown of the end instruments themselves is detectable by comparison with alarm limits, and also by rates of change in excess of those possible with the inertia of the process being measured. The incorporation of self-checking logic is justifiable as insurance against nonobvious error, as an aid to self diagnosis, and as the means for automatically initiating the full diagnostic test routine.

Rapid service, as made possible by automatic detection of error, alarm, and automatic indication of error source, can make the difference between momentary failure and consequential breakdown. When that service can be accomplished by personnel completely unversed in machine logic or circuitry, machine reliability acquires an entirely new definition.

Preventive Maintenance

No breakdown, however, is completely without consequence, even if the consequence is no more than inconvenience to our hypothetical janitor. Detection and correction of incipient failure is always more desirable than correction after failure, no matter how efficiently the means for the latter may be implemented.

Preventive maintenance may eliminate a large percentage of breakdowns, but must be applied with intelligence so that the procedure, in itself, does not set up conditions for other failures. A well-intentioned testing routine, involving periodic removal of packages for analysis, may actually decrease reliability by too frequent stressing of socket contacts. The socket, like every other component, must be given a proper environment for guaranteed longevity. Like the vacuum tube, the socket must not be used if it is to last for the economic life of the equipment. But "not being used," or at least being "used" seldomly, is an economically useful environment for the socket.

Marginal Checking

Marginal checking, as a means for system test under extreme tolerance conditions, may be employed on a data-handling system with considerable advantage and without danger of establishing unfavorable component environments. Built in facilities for varying all system voltages may be used in conjunction with self-diagnostic test routines for rapid location of potential trouble spots. These routines are generally manually applied on a once-a-day or once-a-month basis in normal digital computing systems but can be applied automatically at any desired interval. Applied in this fashion they may eliminate the need for self-checking circuitry. The system is assured, after each test, that it is working properly and has very high probability of remaining so until the next test.

Marginal checking is frequently difficult if not impossible to apply to electromechanical devices, and if a given system requires their use, every effort should be made to obtain a sound basis for life prediction. This requires that a sufficiently large number of the devices must be tested to end of life under conditions which realistically approximate those in the system. It also requires that the components must be manufactured under rigidly controlled factory conditions, so that the statistic may have reasonable accuracy on every such unit in the system. Knowing that a device can be expected to operate without trouble for a given number of years, it is then possible to arrange for replacement at or before that time.

Conclusion

Through proper balance of the logical system, through the use of an intelligently derived modular design philosophy, and through exhaustive analysis of each of the component parts and environments in the system, it is possible to achieve a high degree of potential reliability in an industrial data facility. Present designs should be considered as simplifications of the closed loop system, and make maximum use of digital information handling techniques. With the digital computer, effective system reliability may be substantially improved by built in programs for location of failures in potential trouble areas.

The house of cards has fictitious economic attractiveness over one built of more substantial materials. An industrial data system is very much a part of the factory in which it is used, and as such has an economic value determined over a large number of years. Reliability may justify whatever price is needed for its attainment, but fortuitously it can be shown that the majority of the means are those which actually lead to production economy. In the final analysis, the only sure technique for attaining reliability is a conscientious desire for improvement.

Discussion

D. A. Weir (STL England): Did the speaker say that it was not possible to make marginal checks on electromechanical equipment? Is it common to apply such checks in automatic telephone switching?

Mr. Smith: I believe that the exact words that I used on that subject were, "It is difficult, if not impossible, to apply marginal checking on such apparatus." It is certainly possible to do marginal checking on more elaborate mechanical devices. On simpler devices, such as the socket in which a card is plugged, being a very simple mechanical device, it is rather difficult to do this without shutting the machine down. I think the point here is that we are talking about a completely different order of reliability than heretofore has been considered in the computer equipment. Have any of you, for example, approached the manufacturer and asked for a blower, a simple blower, which he would probably guarantee to operate without any maintenance whatsoever throughout a year? The point is that the way to make an electromechanical component reliable is to use it as little as you possibly can. Marginal checking necessarily implies using it over and above the amount of use that component would receive in the normal operational function of the system. It is to the end of attaining more life out of a component, which has a finite life; and I say that we should not apply marginal checking to electromechanical components.

Accuracy Control in the RCA Bizmac System

I. COHEN[†], J. G. SMITH[†], AND A. M. SPIELBERG[‡]

Introduction

VERY EARLY in the development of the RCA Bizmac system it was recognized that large-scale data-handling applications presented particular requirements which could not be met by the multiplication of data-processing devices alone. The size and complexity of such applications were such that several departures from the organization of earlier data-processing systems were indicated. Centralized and immediate control of the entire system conferred all the advantages on system operation that the programmed computer had on the interim organization of data for computation. All processing machines and data-transfer media not only contain necessary internal monitoring devices, but also the ability to monitor their own system operation.

[†] Radio Corp. of America, Camden, N. J.
[‡] General Electric Co., Phoenix Ariz.; formerly with Radio Corp. of America, Camden, N. J.

Large-Scale System Requirements

The scope of a large data-processing application may be seen from Fig. 1. This represents a daily cycle of operations at the Ordnance Tank and Automotive Command (OTAC) in Detroit, where RCA Bizmac product-line equipment is installed. The data processing encompasses all the bookkeeping functions of OTAC involving stock and supply control, catalog changes, and cross-reference filing, among others, for some two hundred and fifty thousand catalog items. The daily load involves some eighty thousand transactions. At the side of the figure are represented the types of operations that are shown interconnected to form the daily cycle.

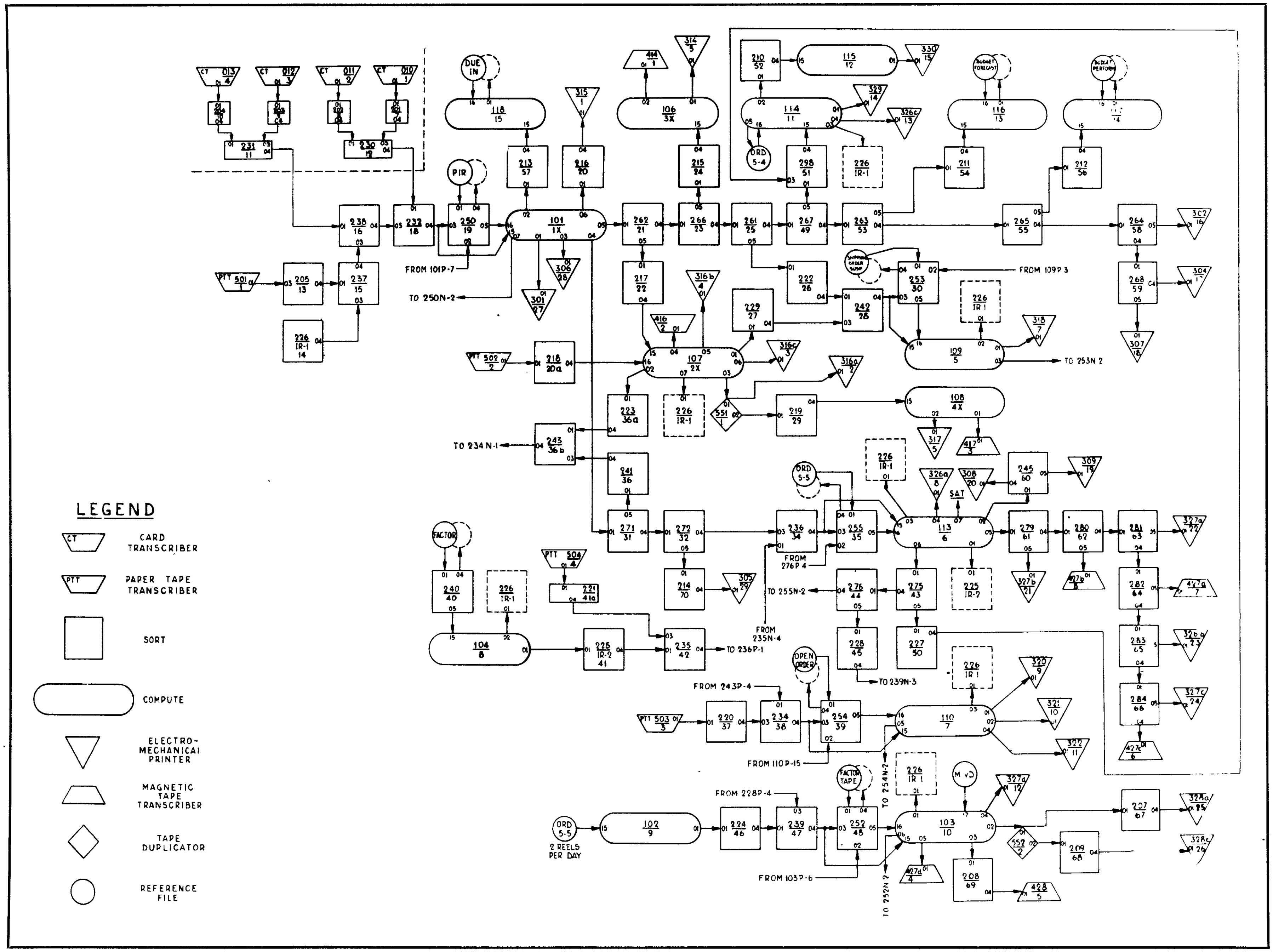


Fig. 1—Consolidated flow chart—one cycle.

The necessity for rigorous data-handling procedures is quite apparent. The variety and sequence of different operations call for the closest attention in verifying the data each machine is handling, the accurate identification of data-storage locations, and the instruction of machines in their tasks. Procedures for organizing the daily routines may be found wanting due to a number of factors. Communication between operators is subject to human frailties. Operator monitoring of operations is not so acute as should be and there are tendencies for watchfulness to decrease with periods of error-free operation. The status and operation of machines would not be known except by constant supervision at their locations. Variations in the schedule due to greater or lesser amounts of daily input or machine down-time would take extended time for modification.

Centralized System Control

The RCA Bizmac System Central was designed to guide the operation of the entire data-processing system. System Central controls extend to the introduction and output of all information to and from the system. It also provides for the semiautomatic set-up, or instruction of all machines, the location of data on magnetic tape and the supervision of all operator's work. Fig. 2 shows the operation of the System Central.

The schedulers provide the sequence of operations for which the system operator selects machines. The system operator then releases the machine instructions, or set-up, to one of the two available operator-verifier teams. The operator and verifier selected then proceed independently of each other (at separate consoles) to instruct the machine and connect it to the assigned tape-files. All Tapefile switching and machine set-up are actuated via the operation control unit. This unit returns a record of all setups and tape connections to the monitor console. It also will guard the system against improper operating routines and operator mistakes by equipment checks and lockouts. Agreements must be reached between the operator and verifier consoles before the schedule can proceed.

The monitor operator reviews all operations prior to their initiation by the system operator and after the operator and verifier have completed a set-up. The monitor console also provides information on the present state of the system. Fig. 3 (opposite) is a photograph of the monitor console. Its upper panel presents the status of all Tapefiles in the system. The lower panel presents the interconnection of Tape Stations and machines. The information gives the particular Tapefiles connected to each of the information trunks of the Computer and Sorters. The schedule number of the particular machine run is shown and, where more than one tape per trunk is necessary, the subrun numbers are also given.

The design of the System Central consoles and the communication procedures used by its operators were "human engineered" by a group of specialists and system engineers. All operational information appears in the form of the actual console controls. The cards bearing the information are color coded for ease in handling.

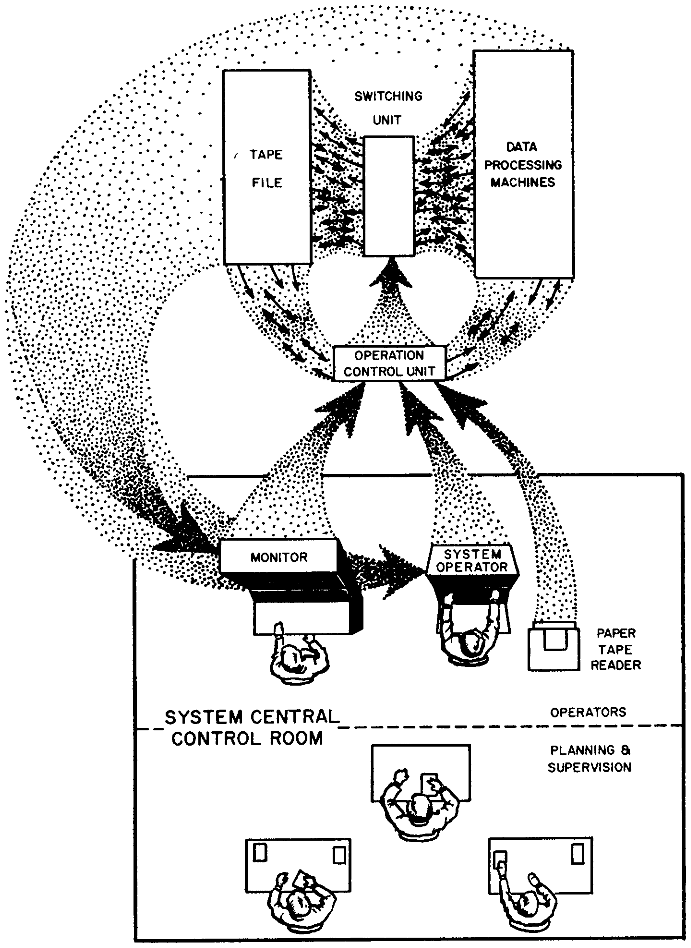


Fig. 2—System Central diagram.

Scheduled operations may also be actuated automatically from paper tape loops containing all setup and tape-connection data for the operation. Every action at the operator verifier consoles is monitored and printed out by a Schedule Recorder for record keeping purposes. An interim scheduler is available to modify the sequence of operations should the need arise. No machine operators can change any data that is being processed by the system.

Data-Transfer Controls

To meet the requirements of rapid availability of frequently used Tapefiles, the RCA Bizmac system adopted the philosophy of Tapes-at-stations. This afforded the system a further advantage in that human intervention was avoided in data-transfer between machines. The System Central classifies Tapefiles into available and non-available for various types of operations so that input and output tapes are not intermixed with the resulting loss of valuable data. When a Tape Station is selected, it identifies itself to the System Central. This identification is in the form of numbers registered beside the Computer or Sorter information trunk to which it is to be connected. Other machines have the same displays at their own console or indication area.

Each Tapefile provides a series of status indicators which may be utilized by the data processing machines connected to it. These are shown in Fig. 4. These status

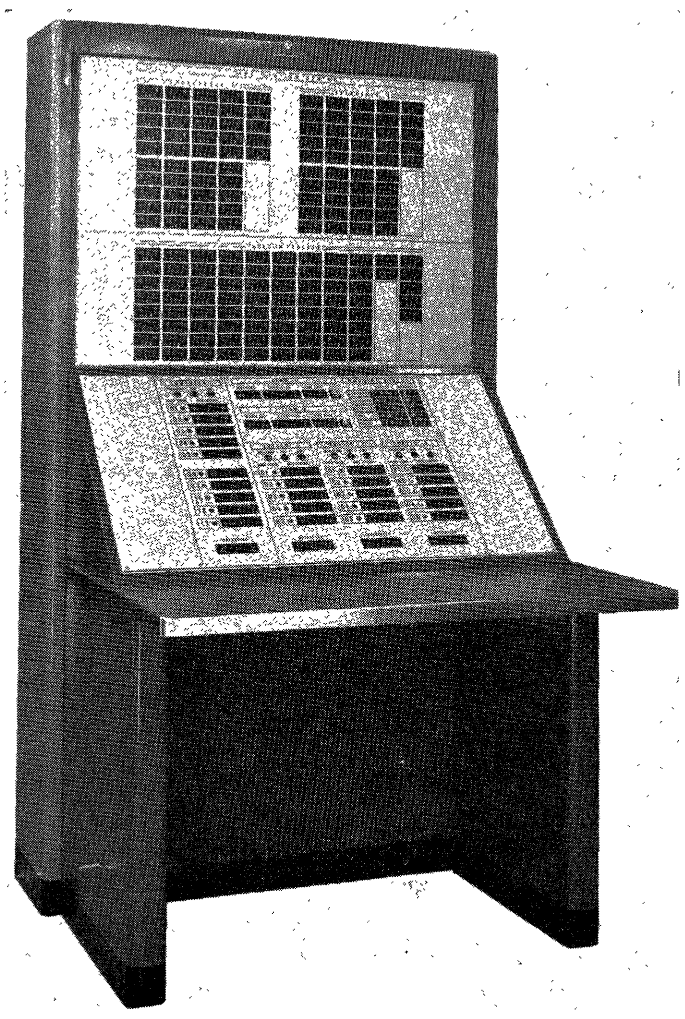


Fig. 3—Monitor console.

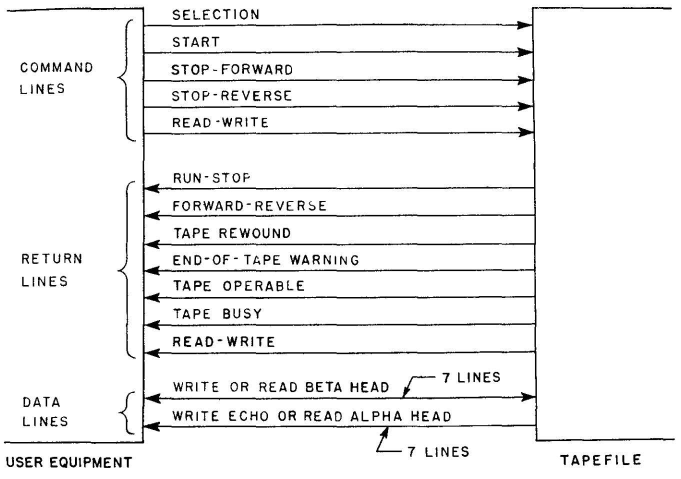


Fig. 4—Control lines for RCA Bizmac Tapefile.

indicators form closed-loop systems which permit Tape Station control completely and automatically by the user machines. All user machine commands are verified. In addition "end-of-tape" warnings and "tape operable" are supplied. The user machines automatically disconnect tapes when they are exhausted and the tapes rewind. To forestall another connection of that rewinding tape for another purpose, a "tape-busy" signal is provided. Upon completion of rewind, a "tape-rewound" signal occurs signifying the tape's availability. Every Tapefile permits checking the current in the writing heads to verify proper transcription. This is termed an echo check.

All machines in the system that work with magnetic tape use Tapefile status indicators. Transcription is monitored by echo checking. Parity checks are made on all data transfer whether to or from magnetic tape or paper tape. Special checks are used in transcribing machines, the Computer and Sorter, to avoid the loss of characters or whole messages. These data-organization checks monitor the sequence of Bizmac editing symbols.

System Equipment Checking

In the system all error-checking devices and procedures are called accuracy controls. The accuracy controls in each area of the system are adapted to the needs of that area. In machines such as the input and output devices, which are generally monitored visually, the functions of the accuracy controls are to call operator attention to errors or difficulties. In machines such as the Computer or Sorters, automatic reruns of the operation are possible so that the machine itself can discriminate between errors due to component failure or incorrect information, and those due to failures of a transient nature such as extreme excursions in line voltage.

The accuracy controls of the system were reviewed during their design to fulfill the requirements of error detection and correction. Error indicators were required to indicate their own failures at the machine console or control area. Error detectors were frequently added to make error location easier and simplify the task of correction. Special detection techniques were used to prevent a machine failure without indication. Comparison devices for duplicated equipments or operations were built into the machines.

The Computer will attempt to rerun an operation in which an error is detected if the error involved is of the nature of incorrect parity, lack of verification, arithmetic overflow or lack of comparison of arithmetic results. All arithmetic operations are performed twice, the second time with complemented operands. However, in cases of hardware failures such as counters or Tapefiles, immediate shutdown results. When the Computer attempts an operation again, the number of retrials is limited by programmed counters. The Computer's programming abilities permit a variety of programmed checks which can only be treated adequately in a separate series of papers.

The Sorter possesses the ability to discriminate between errors causing immediate shutdown and those it can again attempt. As its function is the reorganization of data on magnetic tape it monitors its own use of the Tapefiles.

In addition to the regular tapefile-status indicators which it uses, extra detection devices insure that the proper tapes, and only those, are in motion and reading or writing at any time. Parity checks are provided for all information registers. Message criteria are examined for proper size. To maintain the data rate on tapes special equipment will reconstitute data-pulse densities on tape if it is found to vary considerably from the Tapefile 10-kc rate. Intermessage gaps on tape are also reconstituted if they vary too much from system norms.

Input to the system is by means of paper tape or punched card. Paper tapes are prepared on Tapewriters. They are verified by the Tapewriter-Verifier equipment. The original tape is inserted into the Tapewriter-Verifier and a new tape is prepared by another operator using the original document. Both tapes are compared character for character and their parity checked.

The Card Transcriber transcribes from cards to magnetic tape. Each card is sensed at two separate sensing stations and the data compared. If a lack of comparison exists, all cards in motion are ejected into a hopper for operator inspection. Mechanical checks monitor the card transport and assure the transmission of only one card at a time. In case of error an alarm summons the operator. The Card Transcriber will automatically back up the tape, erase the last message and restore starting conditions.

All input and output devices monitor the condition of the magnetic or paper tapes and cards where they are used. Indications of breaks, jams, low tape supply, or exhaustion are brought to the operator's attention by appropriate alarms. Transcription devices such as the Card Transcriber, Magnetic Tape Transcriber, and Paper Tape Transcriber cannot complete their operation until the operator inserts proper terminal editing symbols. When errors occur in data transmission they all possess the ability to erase improperly transcribed data and restore starting conditions.

The Electro-Mechanical Printer can, on detection of parity errors in data transmission, back up and reread the magnetic tape connected to it before transferring control to its operator. It guards against printed lines of more than one hundred twenty characters. Lack of paper or rips in the paper are detected. The operator can inhibit the parity checkers in the machine to cause the line of data in question to be printed out.

For those machines in the system which do not have the facility to print out the contents of their Tapefiles the RCA Bizmac Interrogation Unit can be quickly connected for printout. It can be made to print out data despite parity errors or improper organization. The Document Printer can be used to page print the contents of any paper tape.

Equipment Reliability

Conservative design techniques are used throughout the system. Circuits will perform their functions if cathode emission drops to one half rated value. Rated dissipations were also cut by one half. The system is fused to retain independence of various areas of the machines and aid in location of faulty components. All component connections are readily available and plug-in, or modular, construction assists rapid repair. Marginal checking equipment is integral to the Computer and Sorter.

Each Tape Station has facilities for marginal checking of magnetic tape and head deterioration. Elapsed-time meters are built into all Tape Stations to record length of tape service. Magnetic heads are constructed to record a character (seven bits) into 14 separate tracks. The arrangement of recording heads are such that bad areas on tape that may develop with time will not interfere with normal data handling. A maximum poor spot of one-eighth inch by five-sixteenths of an inch can be tolerated. This is shown in Fig. 5. All magnetic tape is completely written and checked at twice normal pulse densities before acceptance for system use.

A sampling of the system accuracy controls available are presented here. More detailed treatment of internal and system accuracy controls for each machine can be found in the attached bibliography.

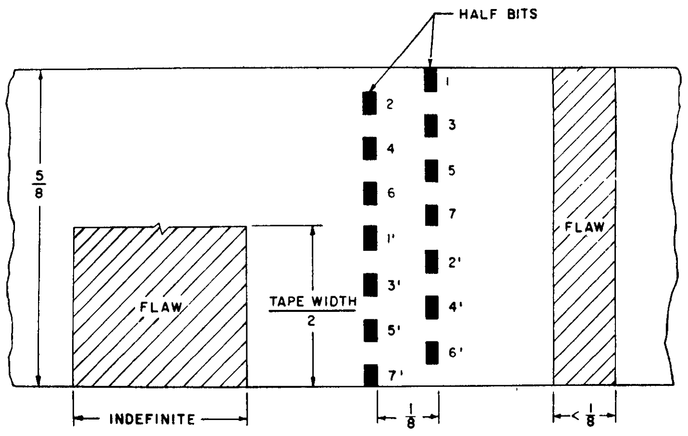


Fig. 5—Seven-bit dual-channel configuration.

Conclusion

The philosophy of system reliability of the RCA Bizmac system is embodied in the concept of centralized control of large scale data processing. Where necessary, the human-machine transfer loop has been strengthened by the development of more potent monitoring devices. A great degree of system flexibility has been achieved. The over-all reliability and efficiency of the entire data handling process has been improved.

Bibliography

[1] Halstead, W. K., Leas, J. W., Marshall, J. N., and Minett, E. E., "Purpose and Application of the RCA BIZMAC System," *Proceedings of the Western Joint Computer Conference, 1956.*

[2] Beard, A. D., Bensky, L. S., Nettleton, D. L., and Poorte, G. E., "Characteristics of the RCA BIZMAC Computer," *Proceedings of the Western Joint Computer Conference, 1956.*

[3] Owings, J. L. "The RCA BIZMAC System Central," *Proceedings of the Western Joint Computer Conference, 1956.*

[4] Beard, A. D., Halstead, W. K., and Page, J. F. "Functional Organization of Data in the RCA BIZMAC System," *Proceedings of the Western Joint Computer Conference, 1956.*

[5] Bensky, L. S., Hurewitz, T. M., Lane, R. A. C., and Kranzley, A. S. "Programming a Variable-Word-Length Computer," *Proceedings of the Western Joint Computer Conference, 1956.*

[6] Montijo, R. E. "Control Philosophy in the Design of the RCA BIZMAC Tapefile," presented at the National Electronics Conference, Chicago, Illinois, October 1, 1956.

[7] Katz, A., Jones, A. G., and Rezek, G. "Accuracy Control Systems for Magnetic-Core Memories," *Proceedings of the Western Joint Computer Conference, 1957* (this issue).

Continuous Computer Operational Reliability

ROBERT D. BRISKMAN†

Introduction

IN THE near future the homes, families, and lives of every individual in the United States will be protected against an aggressor attack by gigantic integrated radar defense networks. The heart of these networks will be large computer systems. With the probable use of ICBM (intercontinental ballistic missiles) and long-range strategic bombers as aggressive weapons, a criterion for defense against surprise attacks is an early-warning capability which must operate 24 hours a day, seven days a week. The consideration of methods to give maximum reliability in the operation of a computer system that must function continuously is of national importance.

Illustrative Computer System

For illustrative purposes, practical discussion will be limited to the typical large computer system described in this paragraph. Extension of consideration to computer systems of different structures or variants would be relatively simple. The computer will be divided into the basic components of synchronizer (clock), memory, logical systems (including program and arithmetic units), and input-output circuits. All operations and data flow will be serial in nature with internal, fixed programming. For ease of maintenance the computer circuits will be modularly constructed in the form of pluggable units. This type of construction will include a ferrite core memory.

Input information for the computer will come from multiple communication sources or data transmission links. Punched cards, punched tape, magnetic drum, or magnetic tape can be used if necessary for intermediate storage. The output data will be both permanently recorded and presented instantaneously on some alerting device for recognition and monitoring purposes. The computer will also have the capability of determining the proper courses of action concerning input and output data flow.

Reliability

No matter what operational characteristics and system organizations are attributed to the illustrative computer, the only important assumption is that the computer system will be so large or complex that a significant number of basic components fail daily. Actually the important fact is not the number of failures but that there is a significant probability of component malfunction.

For the case of a computer system which will continuously operate, the term reliability must be carefully examined. This term was not included in the "IRE Standards."[1] Reliability might be defined in terms of "component failure" which is related to function cessation rather than in terms of "accuracy" which is more related to degree of error. Reliability as concerned in this paper can be defined in relation to the probability of component failure. The degree of success in keeping the computer system continuously operating will be a function of the ability to predict this probability of component failure.

By the addition of a "prediction" function the variable of time is introduced. Prediction can be defined in time for this particular case by four periods.

1) The duration (*dr*) of the prediction interval.
2) The difference (*dt*) between the time of performing the prediction and the time that the prediction interval occurs.
3) The duration (*dl*) necessary for performing the prediction test.
4) The period (*dz*) elapsed between each probability test. For example, the probability of component failure during a one-hour (*dr*) period occurring five hours (*dt*) from performing a prediction test can be determined in a two-second (*dl*) interval which will be repeated every 20 minutes (*dz*).

An improvement in the prediction of reliability may be established by use of a longer *dt* period. However *dt* is determined practically from probability failure curves of various electronic components. These curves are obtained from life testing of the more sensitive electronic devices used in computers such as vacuum tubes, transistors, diodes, ferrite-core drivers, relays, etc.[2] Therefore *dt* will be limited in maximum length to the mean probability of operating life of the components being considered. Likewise, *dr* is limited by similar practical considerations. In general an improvement in the prediction of reliability may also be established by use of a longer *dr* period.

Prediction of operational reliability can be significantly improved by using a higher number of times that probability is sampled per unit time ($1/dz$) or a shorter *dz* period. The restrictions on increasing this probability sampling rate are imposed by the computer system and not by the characteristics of the individual circuit components. Basically, one restriction stems from the fact that any time used for probability sampling in a continuously operating system must be considered computer down time. High down time in a computer system creates a poor duty cycle and a low operational efficiency. Also, a normal requirement is that the computer

† Army Security Agency, Arlington 12, Va.

[1] "IRE standards on electronic computers: definitions of terms, 1956," Proc. IRE, vol. 44, pp. 1166–1173; September, 1956.

[2] For further discussion of this topic see IRE Trans., vol. EC-5, no. 4; December, 1956.

output devices will need a certain information flow per unit time to properly function. Therefore the probability sampling rate is secondly restricted to low enough values that the information output flow necessary for continuity will be sufficient. The time required for performing each probability test (*dl*) should be as short as possible to allow either a higher number of probability samples or a decreased amount of down time with a corresponding increase in the output information rate. Actually *dl* is equal to the down time per operational period and usually is inversely proportional to the probability sampling rate.

From the above, one justified approach to maximum reliability in a continuously operating computer system is to attempt recurrent prediction of values for the precise probability of component failure during a specified period of time in the future.

Prediction Testing

Prediction testing on computers is a commonplace practice today. However, almost no computer systems must face the stringent requirements of continuous operation and prohibition of long-term outages.

The most common prediction test for computers is some type of diagnostic programming or programmed checks. A diagnostic program is designed to test as many circuit components as possible throughout the computer system in a rigorous manner to obtain early indications of malfunctioning parts. These tests are often performed with increased or reduced operating voltages. The variations in operating voltages are designed to place sensitive circuit components nearer a marginal region of performance. This causes the malfunction of components with the higher probabilities of failure. By replacing these components the over-all computer's *dt* and *dr* periods are increased for the subsequent operating period. For a certain large business computer which operates eight hours daily, one diagnostic program a day insures a 90 per cent reliability during the operating period. The whole diagnostic program takes 40 minutes including test runs with variations of plus and minus 12 per cent in all dc operating voltages.

Several disadvantages are apparent when diagnostic programming of this type is applied to a continuously operating computer system. The major disadvantage is that the operating section of the computer must usually be halted to perform the diagnostic program. As a thorough diagnostic test will probably take considerable time, it is necessary to switch to an auxiliary operating section of the computer. Switching of computer sections is always necessary if variations in operating voltages are used as part of the diagnostic testing.

The actual switching from one operational section of a computer to an auxiliary section will normally involve significant down time when the memory blocks are considered. If the data in the operational memory is unique, this information must be transferred to the auxiliary memory section with accompanying loss of time. If the data is accumulative from experience or computation, either the transfer can be effected or new data can be regenerated. Both alternatives may result in significant computer down time and may possibly cause a short term output data flow rate below requirements.

To illustrate the above, a transfer of information from the operational memory to an auxiliary memory by use of the computer's logical system would involve a minimum instructional program of SELECT, LOAD, SHIFT, SELECT, STORE, TRANSFER plus necessary check cycles. The actual time needed for a transfer can easily be computed from knowledge of storage capacity of the memory, time required for operational cycles, number of operational cycles necessary per transfer, and amount of information that can be transferred per operation cycle (usually limited by the accumulator length). With coincidence-driven ferrite-core memories in the order of 10^9 bits of storage, the time lost using the computer logic system for a memory transfer operation will be appreciable. On the other hand, if the auxiliary memory must be regenerated, even greater time can be lost when pertinent data is obtained through extensive recomputation (*e.g.*, a memory containing one thousand targets all stored in vectorially computed positions).

Computer System Organizations

It has been noted that a continuously operating computer system may consist of one or more integral computers. Obviously when failure during operation occurs or when normal maintenance is necessary, at least one stand-by computer unit would be required to insure continuity of operation. More elaboration on some possible configurations of computer systems is pertinent.

An obvious configuration is a system composed of three separate computers. One is in continuous operation, the second in stand-by, and the third in maintenance or semistand-by. When the computer in use makes a certain number of mistakes uncorrectable by the programmed error correction routines or exhibits operational errors through other failure detection devices, operations are switched to the stand-by computer. The stand-by computer has been previously prediction tested so extremely reliable operation will be maintained while the operational computer is in repair. After maintenance, the operational computer can be prediction tested and then become the stand-by computer. If the stand-by computer does fail during the time the operational computer is under maintenance, a third computer is still available.

Advantages of using the three-computer configuration are that individual computer down time is relatively unimportant and that no limitations are placed on the long term continuity of data output. However, computer system down time and short-term discontinuities in output data flow will be determined by the time required to transfer or regenerate memory information. Although this configuration for a continuous operating computer system obtains an exceptionally high reliability by a brute force method, there are the previously mentioned advantages in such a solution. These were accrued by effectively reducing the problem to one computer operating on an eight-hour shift, substantially lowering the over-all efficiency of such a system. No further consideration will be given to a three-computer system due to the prohibitive construction cost.

A second configuration is a system composed of two computers. Although the cost of such a computer system is extremely high, it is the minimum construction with which any type of continuity of operation can be maintained.

One variation of the two-computer system configuration involves the use of an operational and a stand-by unit. The method of employment would be exactly similar to the three-computer system arrangement previously described, with the computers automatically switching upon failure of a significant component. The obvious objection to this system is the possibility of failure in the second computer while the first is being maintained, resulting in complete cessation of data output. However a computer simple enough for rapid maintenance and with an extremely low probability figure of component failure could efficiently operate in such a configuration. All the advantages and disadvantages of a two-computer system will be similar to the three-computer system with the above noted exception. To obtain maximum reliability with this two-computer system, the stand-by computer can be subjected to various prediction tests. The prediction tests should establish an insignificant probability of component malfunction over the time interval determined necessary to service the operational computer (normal or failure type of maintenance).

A second variation of the two-computer system configuration involves prediction of a predetermined high reliability figure for each of the computers over a specified time interval of practical length. At the end of this period the computers are automatically switched. For example, assume computer one is given stringent diagnostic programs until a prediction of 98 per cent reliable operation can be obtained for a certain minimum period (*e.g.*, three hours). Operations are started with computer one. Meanwhile computer two is subjected to the same diagnostic tests until it can be predicted that this computer can operate with 98 per cent reliability for the established minimum period. If computer one does not fail during the predicted three-hour reliability period, the computers will be switched at the end of that time. Failure of computer one during the predicted reliability period will cause immediate switching of operations to computer two. Computer one will then be resubjected to the prediction tests and become stand-by until computer two either fails or completes its predicted reliability period. The two computers are then alternated based on the above criteria to obtain continuity of operation.

Three of the major advantages of this computer configuration are listed below.

1) The over-all operational reliability of the above computer system can theoretically be made extremely high.
2) The computer system down time and short-term discontinuity in output data flow is determined solely by the switching time necessary to transfer stored information.
3) The majority of the switching operations normally occur when the operating computer is in perfect working order. In all previous cases this information transfer must be attempted with a malfunctioning component in the computer which may nullify the applicability of the transferred data.

One of the major disadvantages of this system is that a large number of switching operations will be necessary. This high switching rate will be required to maintain a prediction of low probability of component failure (*i.e.*, a high reliability). A high probability of component failure is inherent due to the large number of sensitive components used in present day computer circuitry. The ability to predict high reliability figures for practical periods depends directly on the mean life of the more sensitive circuit components. These mean life spans are relatively short for an application in this computer configuration. Another disadvantage is that this configuration is subject to complete stoppage if both computers fail consecutively or during a short interval.

If this system is to be practical, the time between each switching must be long. Also, the time required for performing the prediction test must be much shorter than the nominal predicted reliability period. Using the above computer configuration, transfer of information in memory cannot be done through the logical circuitry as too much time will be lost. A direct method of data transfer between memories must be used despite additional costs.

Continuous Internal Prediction

Of all the configurations previously described for a continuously operating computer system, the second variation of the two-computer system seems to utilize best the benefits of prediction and to indicate a direction towards solution.

One of the disadvantages of all the previously-mentioned computer systems is that prediction testing of the operational computer can only be accomplished before or after operation. Therefore no prediction testing can be done until either computer failure has occurred or a specified time interval of relatively long duration has elapsed.

A configuration is proposed as a possible solution whereby prediction testing is accomplished as an independent part of each operation cycle or multiple thereof. The use of prediction testing of the type previously described must be discarded as the time for the prediction test must be extremely small compared to the computer's operating cycle. This is necessary to preserve a workable duty ratio. If a recurrent prediction test substantially increases the computer duty ratio, the probability of component failure will significantly increase with a corresponding decrease in reliability of operation. With present day computers, a 10-microsecond period for a prediction test (including circuit recovery time) repeated every 50 operational cycles may be a high figure. Variation of operating voltages as an aid to prediction testing for these short periods would also be impossible. Therefore a totally new method of prediction testing must be established.

The first consideration in developing a prediction test would be to establish new standards for component failure probability. This prediction test will be based on the application of an internally generated waveform to the computer circuitry. Each modular unit will be broken down into the various component circuits such as triggers, inverters, gates, amplifiers, delays, couplers, etc. These circuits are designed for normal operation with certain minimum-pulse type inputs. Therefore, life testing of these component circuits with various degenerative varieties of input test pulses will be performed in order to establish a favorable prediction test. A favorable prediction test would be defined by that test waveform input which causes the greatest number of malfunctions in those circuit components with the high probabilities of failure.

The parameters of a pulsed test input waveform that can be easily varied are the amplitude, shape, width, and recurrence frequency. The single test input which will provide the most favorable prediction for all the computer circuits will be a composite of pulse parameters. For example, an amplifier might respond primarily to variations in pulse amplitude, diode gates to pulse shapes, filters to pulse width, and multivibrators to pulse recurrence periods (especially transistorized triggers). A possible test input which might give favorable prediction would be a double pulsed waveform. This waveform would consist of two test pulses, each being one-quarter to three-quarters the width of the narrowest pulse used in normal operation. The two test pulses would be separated by a definite time period less than the shortest operating pulse spacing and have an amplitude of one-half to nine-tenths of the minimum voltage levels used in design. Various rise and fall times of either or both test pulses would also be employed to obtain the best prediction test. The entire duration of the test input waveform normally should occupy a period less than 2.0 per cent of an operation cycle. This would include circuit recovery time.

A computer configuration using a prediction testing system similar to that proposed above would not be very costly. The diagnostic circuitry would consist of the test waveform generator, an elementary timer (to synchronize the prediction test with the operation cycle), line drivers, and failure detection circuits. Each pluggable unit or modular circuit will also require a test waveform input and an error detection output. The computer system may employ separate busses for conveying the test input waveforms to the various sections of the computer.

The failure detection circuits can be built into each computer block , each operating section, each pluggable unit, or each modular circuit. Some advantages of placing failure detection devices into as small an operating entity as possible are listed below.

1) Failure location is tremendously simplified.
2) The actual detection devices are extremely simple.
3) The probability of error in the failure detection circuits themselves is reduced.

The above advantages may be nullified if it is necessary to use such a large number of simple detection devices that computer size and costs are substantially increased.

It is obvious that a computer system employing continuous prediction testing as an integral part of the operational cycle must be designed around the prediction-testing technique. This proposed type of testing cannot be easily added to existing computer systems. The computer system should also be designed with circuitry to allow complete transfer of information contained in memory to an auxiliary memory without the use of the logical circuitry. The construction needed for such a parallel-type transfer may be quite expensive although simple in design and extremely rapid in operation. In practice it may be advantageous costwise to sectionalize the memory and provide spare memory sections in case of failure in operating sections. Although this avoids fabrication of two complete memory blocks in parallel, sectionalization will involve considerable additional circuitry for addressing, reading, writing, parallel switching to auxiliary sections, and failure detection in each separate memory division.

Conclusion

Reliability for a continuous operating computer system can be considered by many approaches. In general, these computer systems are composed of multiple duplicated units which are interchanged to maintain continuity of operation. Of the approaches considered, a computer system which incorporates continuous prediction testing as part of the operational cycle seems a promising solution. Using this approach for design of the computer configuration and component circuits, continuous operation of a computer system may be obtained with high reliability by establishment of a favorable prediction-testing method. Computer design will also include the necessary circuitry for direct memory transfer upon failure detection.

The computer system using the proposed prediction testing technique would have the advantages of insignificant long and short term discontinuities in the information output, a low data error level, negligible system down time, a high efficiency, and relatively low costs. The proposed prediction testing method is also advantageous as each memory transfer would be accomplished under normal operating environment, while the component failure had occurred during the more stringent test conditions.

Bibliography

[1] Richards, R. K. *Arithmetic Operations in Digital Computers'* New York: D. Van Nostrand Co., Inc., 1955.
[2] Shea, R. F. *Principles of Transistor Circuits,* New York: John Wiley and Sons, 1953.
[3] Goldman, S. *Information Theory,* New York: Prentice-Hall, Inc., 1953.
[4] Engineering Research Associates. *High Speed Computing Devices,* New York: McGraw-Hill Book Co. Inc., 1950.

Discussion

Chairman Parsons: The speaker was very careful to develop a theoretical computing machine for his description of the continuous computer operational reliability; I wonder if we can comment on it concerning the applicability of this particular marginal technique to such existing systems as the SAGE?

Mr. Briskman: I stated, I believe, in the close of my speech that this cannot be added to the existing systems. My familiarity with the SAGE system is very slight; however, from the size there it might have been a very interesting experiment to try the prediction system of techniques, such as was proposed, rather than the standard formula technique which is employed. In other words, reducing various dc operating voltages, and running diagnostic programs. Actually the SAGE system is probably a little small to benefit on a cost basis, on changing to this alternative method of prediction testing.

J. G. Tryon (Bell Telephone Labs.): I question whether continuous internal prediction testing via special test signals can be realized with moderate equipment cost. Restoration of wave shapes is so extensive in a good computer that very many test signals and associated verification circuits would be required. I estimate that the size of the computer would be doubled.

Field Performance of a New Automatic Fault-Locating Means

J. F. SCULLY† AND L. P. COLANGELO‡

A MODERN Air Force electronic system does the work of scores of people. Viewed as a labor-saving device, it is comparable to an automatic telephone exchange. There is, however, a great organizational difference between these two automatic systems. A digital calculator, for example, is essentially a single unit of great complexity, whereas the telephone exchange is composed of a multiplicity of units, each capable of working independently. The telephone exchange can be operated successfully by disconnecting its malfunctioning circuits, but the entire digital calculator is rendered useless if one component fails. It is as though all the automatic clerks have staged a walk-out until the single troublesome source is located.

The net result of the ever-increasing complexity of electronic equipment has been that the shortage of adequately trained personnel in the Air Force has been accentuated. Not only has research and development work been hampered, but the reliability of operational field equipment and the establishment of sound maintenance programs for such equipment, have been adversely affected. It has therefore become incumbent upon designers, engineers, and manufacturers to strive for greater simplicity of electronic equipment and to produce equipment easier to maintain.

At the outset, we must clarify what we mean when referring to "reliability" in connection with a large ground electronic equipment. Since such equipment is repaired, and so made operational again after each failure, it is a different problem from, say, a missile or system in which one failure renders the device useless for all time. So, while the mean time between failures is of great importance, and the probability of successful operation for a given time interval is also important, an additional factor, the "down time" of the system, is of equal importance. We shall define the reliability of our system in terms of its operational efficiency as follows. The efficiency of a system is the ratio of the time during which the machine is capable of correct operation to the time during which correct operation is desired. Thus, if correct operation is experienced whenever we want it, the system has an efficiency of 1; if it never works when we want it to, an efficiency of 0. This definition makes

† Monroe Calculating Machine Co., Morris Plains, N. J.
‡ Rome Air Dev. Center, Rome, N. Y.

it easy to examine the characteristics of a machine of given efficiency in terms of ease of maintenance for that machine. Table I presents the mean trouble duration per trouble which would cause the observed efficiency if the number of troubles encountered during an operational week of 168 hours were experienced.

TABLE I
Down Time vs Efficiency

Troubles per Week	Efficiency		
	0.75	0.85	0.95
	Trouble Duration (Hours:Minutes)		
1	42:00	25:00	8:00
2	21:00	12:00	4:00
4	10:00	6:00	2:00
8	5:00	3:00	1:00
12	3:30	2:00	0:40
25	1:40	1:00	0:20
33	1:15	0:45	0:15
50	0:50	0:30	0:10
100	0:25	0:15	0:05

This table suggests that electronic machines are beset by serious problems. For any fixed efficiency, if there are very few troubles, then each must be very difficult to locate. Let us suppose that every endeavor has been made by a manufacturer to reduce the occurrence of faults to the lowest possible number with all the techniques at his command. When all advantage possible has been taken in this direction, the system will exhibit an efficiency which is now a direct function of the time which it takes the maintenance personnel to remove the troubles which occur.

An automatic fault-locating means has been devised, reduced to practice, and subsequently utilized in a large-scale digital data-processing equipment built for the Rome Air Development Center by the Monroe Calculating Machine Company. The machine solves a classified Air Force problem and contains several thousand logical elements (vacuum tubes and diodes). The Monrobot Automatic Internal Diagnosis (MAID) monitors the machine at all times to make sure that there are no circuit failures which might cause errors in the solution of the problem. Upon occurrence of a failure, the diagnosis unit quickly and automatically localizes the circuit at fault.

The genesis of this automatic fault-location system lay in the answer to the question: since electronic calculators have been designed to perform automatically operations otherwise handled by humans, why do we not design maintenance machines to replace servicemen? MAID is a pioneer answer to this question.

Selfrepair implies that maintenance operations are completely mechanized. Five steps are required where maintenance is to be done on electronic calculating or switching circuits: error detection, fault location, component replacement, error clearing, and restarting. With the exception of component replacement, each step is easily susceptible of mechanization. The system here considered mechanizes these steps and so greatly reduces the time required to repair a fault when it occurs. It does so by scanning points of possible error in the machine. If one point exhibits defective behavior earlier than another, it is regarded as a "better" cause of the error than the other; if, on the other hand, two points exhibit the error simultaneously, that point which is functionally independent of the other is taken as the "better" cause. When all points of the machine have been scanned, one point stands out as the cause of all of the errors which were observed during the scanning process. Phrased alternatively, we may say that of all points of possible error, some subset of points will exhibit the error at the earliest time; of this subset, that point which is functionally least dependent on the others locates the cause. A decimal number assigned to this point appears in lights on the control console; replacement of the plug-in unit associated with this number by a spare completes the repair and permits correct operation to continue.

The digital data processor built for the Air Force was selected for the prototype application of MAID shortly after a demonstration unit, which proved out the basic feasibility of the particular approach used, had been completed and operated successfully in the laboratory. Consultation with Rome Air Development Center representatives revealed that such a system was greatly desired. Design of the automatic fault locator proceeded in parallel with design of the data processor itself.

Because of certain technical uncertainties at the time construction of the machine was begun, it was decided to apply the MAID to the logical section only of the machine. It was realized at the time that the exclusion of other sections from the diagnosis would, of course, make those sections of the equipment more difficult to trouble-shoot than would otherwise be the case. On the other hand, since the machine in question was an experimental model, it was decided that the advantages of full application could more efficiently be made in later machines after the technical difficulties had been removed. (As of this writing, several machines, using the MAID throughout, have been delivered to customers and successfully operated in the field.)

The data processor was delivered to the Electronic Warfare Laboratory of Rome Air Development Center on September 1, 1955. Installation was completed, and power applied on September 2. The equipment was correctly processing test data the same day. There followed a two-week period of acceptance testing by Rome Air Development Center Engineers. In the period that followed the conclusion of these tests, a detailed operational log was kept to provide a complete picture of the efficiency and maintainability of the machine. During this period, 360 hours of operation were scheduled, the equivalent of nine 40-hour work weeks. Of this time, 13 down-time hours occurred, so that the operating efficiency of the machine was 0.96 over-all.

The 13 hours of down time were brought about by 20 separate faults. Of these, 8 were in the sections of the machine with which MAID was integrated and accounted for three hours total down time, an average of 22 minutes for each fault. The remaining 12 occurred in sections to which automatic diagnosis was not applied, and account for 10 hours, an average of 50 minutes each. Thus it is apparent that MAID effected a marked decrease in the down time per trouble as encountered in actual field experience, since the same maintenance personnel serviced the machine over the entire period. In actuality, the results of using MAID were even more favorable than these figures indicate, since it was decided at the outset of the operational logging period to maintain the log in 15 minute quanta only, thereby making it impossible for any trouble to appear in the log as requiring less than 15 minutes. Six of the eight troubles diagnosed by MAID are logged at this minimum interval; hence, they actually required less than 15 minutes each. Only two of the 12 troubles not in the section diagnosed by MAID occupied this short a time.

Since the MAID was applied in this pioneer equipment to about half of the data processor, conclusions can be formed as to its effect on the efficiency of that part of the equipment as compared to the part not having this advantage. One may suppose that, on the average, about an equal number of troubles would occur in both parts. (The actual poorer experience with respect to number of troubles in the nondiagnosed section was in all likelihood due to chance.) Thus, had the entire unit been diagnosed by MAID, the expected efficiency may be computed by assuming that 20 troubles would have occurred each requiring 22 minutes or less to repair; this machine would have at least 0.98 efficiency. Its counterpart, the data processor with no automatic fault location, would have had 20 faults requiring, on the average, 50 minutes each; this corresponds to 0.95 efficiency.

The automatic diagnosis means employed has additional advantages not reflected directly in the efficiency figures. One of the most important of these is the reduction in the level of training required to maintain the equipment. Since the difficult task of locating troubles is now mechanized, most troubles can be serviced by personnel with very little technical skill who could not, without this aid, service the equipment at all. Secondly, as a production trouble-shooting means, impressive reductions in delivery schedules can be made with no increase in technical staff, a matter of great importance in these times of critical shortage. Thirdly, training of personnel to a high degree of efficiency is made much easier, as the machine itself does much of the teaching! Also of great importance is the security inherent in the certain knowledge that an equipment is performing correctly at a given time.

The application of automatic diagnosis discussed here is the first in the art. We believe that it has fully lived up to expectations, but recognize at the same time that improvement is certainly possible. In the first place, the desirability of extension of the principle to include complete systems, rather than portions only, was clear from the outset. The practical limitations which weighed against doing this in the prototype have since been removed and succeeding calculators have been built with the completely automatic system. The advantages of doing this have been realized in practice. Also, it has been recognized that MAID application should be considered as a vital part of the design of equipment if its full potentialities are to be achieved. Experience has shown that applique construction is often possible, but not as efficient. Again, since the past experience of the people concerned has been largely in the field of digital techniques, it was natural to devote the major application effort in this field; however, it must be stressed that the means are applicable to many nondigital electronic systems. Work is progressing to improve the actual means employed so that both simpler circuitry and even more rapid fault analysis can be made. Consideration has also been given to more direct correlation of trouble indication with defective component to still further simplify the duties of maintenance personnel.

The future of large electronic systems depends to a large degree on the ability of those of us in the services and industry to break through the barriers of maintainability and reliability which still lie ahead. Components people are working ceaselessly to provide better pieces for equipment people to use; surely systems designers can do no less. It is perhaps, then, not inconceivable that the day will come of which we all dream from time to time—the advent of the perfect electronic system!

Discussion

N. J. Dean (Ramo-Wooldridge): How much do the dual elements add to cost? How much does MAID add to cost? Would you dualize storage units?

Mr. Scully: If you are using the system for production line debugging, the dual units cost two or three per cent more for connections to the existing device. If you are going to do your testing by one equipment with another unit alongside, plugging the two together trouble-shooting does not cost anything (SIC). If there are dual equipments anyway, as in SAGE, because of military necessity, there is no additional cost either.

If, on the other hand, it is a system where you can afford to have the down time, as might be the case in some applications, the cost will be 100 per cent of the extra equipment, and the expenditure probably would not be justified.

The MAID unit istelf, once we suppose the means for determining whether a given point is performing properly or not, as exemplified by dual equipments, it is probably only a matter of some five to ten per cent of the cost of the unit.

Yes, we dualize storage units. That does not necessarily mean in the drum system;

for example, one must provide two drums. One might have one drum which has duplicate heads, and duplicate tracks, without two separate rotating mechanisms.

J. G. Tryon (Bell Telephone Labs.): How does diagnostic equipment step from point to point when trouble is observed? How much equipment is required for the MAID scheme?

Mr. Scully: The presence of an error is detected by a comparator unit which initially scans a critical point, or points, of the dual equipment. The occurrence of an error results in the setting of a flip-flop, and that flip-flop holds the error in condition until the necessary control logic has caused the scanning mechanism to take one step. Actually it is wired into cables which terminate at the side of the cabinet and the equipment. The MAID unit is plugged into these, so that each point has one wire which carries it over to the central MAID circuitry. So that in the models which are now in the field, the scanning process is accomplished by step switches. This is not necessarily the only means, but it seems about the cheapest at the present state of the art.

D. A. Weir (STL England): Has the connection of the testing equipment any deleterious effect upon the equipment under test due to its great capacity, or circuit loading? To what digital rate equipment has the MAID ever applied?

Mr. Scully: The connection to the equipment is one of the problems where you have to face the extra capacitances that are introduced. It so happens that the circuits to which the unit has been applied in our own production are such that the capacitances introduced are negligible.

The maximum rate to which it has been applied to date in existing equipment has been 120 kc; however, that is the clock rate on the equipment on which it is operated. The time which it takes the MAID unit to take one step in the diagnostic process is essentially a function of the scanning mechanism, which being step switches, are confined to about 50 steps a second, maximum. The electronic portion, which does the error detection, and makes the decision earliest in time, of course, can be made to operate on equipments essentially as fast as one pleases, taking into account cabling problems.

The Variable Word and Record Length and the Combined Record Approach on Electronic Data-Processing Systems

NEAL J. DEAN†

IN THE literature concerning electronic data-processing systems there has been much discussion of the advantages of the variable word-length feature as opposed to the fixed word-length restriction. It might be well at the outset to specify what is meant by a "word." A word is defined by the IRE Committee as "an ordered set of symbols which is the normal unit in which information may be stored, transmitted, or operated upon within the computer." It is characterized by the fact that it is usually a single unit of information about the record, such as the "balance on hand" in an inventory application or the employee's current weekly salary in payroll. In some cases it should be pointed out that this restriction does not strictly apply, because there can be a combination of several independent items of information in a single word. This might be referred to as a hybrid word and is frequently resorted to in order to increase the efficiency of storage, when the individual items are short—such as a yes-no condition. However, this is an exception and in general the comments made will apply even when this hybrid technique is used.

The Word-Length Problem

If a machine utilizes a fixed word length, it means that all of the items of information within a record and from record to record must be of the same size. This is a rather stringent restriction and one which, in general, results in wasted storage space. Consider, for example, the restriction applied to a payroll application where in a given employee's records is stored several items of information including his annual salary and his hourly rate. In a typical case, the annual salary may require six or seven decimal digits (including the cents digits) and the hourly rate would be typically three decimal digits. If the same size word had to be used for both of these items of information, this word length would have to be at least seven decimal digits long. If it were seven decimal digits, the hourly salary would be using less than one-half of the assigned space. Hence, we see a relatively inefficient storage situation resulting from the fixed word length.

We might also consider the variation of word length for the same word from record to record. Of course, if all of the words within a machine were a fixed word length there would be the same space penalty, not only within the record, but also from record to record. However, there is a degree of variability which has been built into some commercial machines which consists of the following: the individual words within a record can be of different size but must be preset by the programmer for a given application. Then they must be of the same size from record to record. For example, if the annual salary for an individual were word number one and it were assigned seven decimal digits in Record 1 (for a certain employee), it would have to be seven

† Ramo-Wooldridge Corp., Los Angeles, Calif.

decimal digits for every record (every employee). This would not result in as severe a loss in storage efficiency, however, since the degree of variability for a given word from individual to individual is not as great (probably only the variation from seven to six decimal digits for annual salary). Similarly the hourly rate word might be assigned three decimal digits which would probably accommodate the entire range involved in the payroll.

The Record-Length Problem

Now let us turn our attention to the variability in a *record* size. A record might be defined as all of the individual items of information (or words) about a given file unit (for example, the employee in a payroll application, the part number in an inventory application, the depositor's account information in a commercial check-handling application for a bank, etc.).[1] It is obvious the degree of variability in a record can be greater than that in a word since it can vary not only in the length of the individual words but also in the number of words that make up the record. The latter variability can be a much more serious one even than the variability of the word length, in cases where individual transaction detail is to be stored on an account.

For example, in a commercial deposit-accounting application in a bank, the number of checks drawn on a given account in a given month, may vary from tens of thousands for a large corporate payroll account to even zero for some individuals' accounts or inactive business accounts (where the business restricts the use of the account to rare entries). In fact, there are quite a few "dormant" accounts in the typical system, which have *no* activity month after month.

Obviously if a record of fixed size were to be assigned in the electronic data-processing system to accommodate all of the depositor's accounts for the commercial bank, it would either be much too large for the inactive accounts resulting in a ridiculous waste in storage space or the more active accounts would exceed the capacity assigned and would overflow. One might now consider assigning different length records to the different accounts based upon past experience or predicted activities. This would certainly result in increased efficiency; but there is also the degree of variability from month to month for a given account. Therefore, even if this technique of assigning a fixed space dependent upon past experience with an account is used, either a much larger capacity than the account needs on the average would have to be assigned or the frequency of overflow would be large. In addition, the procedure involved in assigning a specified space to each individual account in a commercial bank may prove quite unwieldy. This is particularly so, since the bank in general is not aware of how active an account will be when it opens, indeed, the individual depositors may not accurately know—particularly where there are several special purpose accounts for a business. The controller's office for that business frequently shifts the significance of these accounts and the activity on the individual accounts changes radically.

Variable vs Adjustable

Thus, we can see from the above discussion that there is a tremendous advantage in at least being able to *set* a word length and a record length in advance to different sizes depending upon the application. Preferably, this should not be referred to as a technique of "variable" word and record length, but of "adjustable" word and record length. The word and record lengths are set in advance, not necessarily all the same, of course, but of a length which must persist throughout the application. In the case of word lengths, they must be the same for the same word from record to record. In the case of the records, the individual record lengths have been preset and must maintain this length during the operating period.

However, even this restricted degree of variability which we have referred to as "adjustability" is of great value in improving the efficiency of storage as we can easily see considering the above two examples of the variable word length (between the annual salary and the hourly rate) and the variable record length (between active corporate accounts and inactive individual accounts). In fact the casual observer might feel that the combination of the features of adjustable record and the adjustable word lengths goes so far toward optimizing record storage efficiency that it would be adequate.

However, let us consider in a little more detail the commercial banking application. The specific dollar amount on individual checks might conceivably vary from even one or two digits to a maximum (in regular checking accounts for commercial banks) of about 10 decimal digits. A study made by the author in a large commercial bank indicated that the *average* was about 4.5 decimal digits. Hence, if the word length—even if adjustable—assigned to each individual item was 10 decimal digits, the efficiency of storage for this information would be less than 50 per cent. Since the programmer or system designer will not know in advance how long the individual transactions charged against the account will be, the adjustable feature is of no assistance in reducing this waste storage space. If, however, the system were able to accommodate a truly "variable" word length in which the individual items would be only as long as required to store the information in the item and would be placed densely in the storage space, then a truly efficient storage system would result.[2]

[1] A record is defined by the IRE Committee as "a unit of correlated information relating to a single person or article." However, in many machines a record refers to the largest block of information which can be directly transferred as a unit.

[2] This has some important implications for an addressing scheme which is beyond the scope of this paper to discuss, but those readers who are familiar with the problem will recognize it as requiring a technique of word addressing rather than character addressing for locating information in storage.

"Expandable" Record Lengths

If the data-processing system, then, accommodates a truly variable word length, the loss in efficiency that would result from requiring that each item have the same word length would be eliminated. However, the wasted space due to the fact that the record length cannot be predicted in advance would still remain. If the record length were adjustable and set on the basis of experience, the wasted space would only be that resulting from fluctuating activity from month to month, but that can be quite large. There is a technique which can eliminate even the wasted space due to the variation in record length from month to month. This technique might be referred to as an "expandable" record length in which there is no fixed space assigned to the record but the entire file of records is constantly rewritten whenever the file is updated. This might be likened to an expandable file drawer whereby the space assigned to any particular account is truly expandable and simply pushes back the rear end of the drawer when necessary.

This system is actually afforded in most magnetic-tape file systems; where the record length is not fixed and where the entire tape file is updated; the entire file is rewritten on an output tape and the new items to be entered are merely inserted into the individual account storage and written together with the previously accumulated file for each account on the output tape. Thus, we have an expanding tape file as new activity is introduced. In the case of the deposit accounting application for a bank, the tape-file length would be a minimum at the beginning of the month and expand to a maximum at the end of the month. Presumably at that time the conventional printed statements would be prepared, and the magnetic-tape file for that account wiped clean with the new balance being that obtaining at the end of the just concluded month.

Combined-Record Technique

Certain types of storage media do not lend themselves to this technique of expanding record length. They have, however, other operational advantages which sometimes make it desirable to incorporate these media in a data-processing system. For example, a magnetic-drum file would normally not be rewritten during each processing, since one of the advantages is that of random access and only the accounts which have been active need to be posted. This reduces the time for updating the file and makes it more feasible to post activity in an "on-line" fashion, in random, and more promptly. In a magnetic-tape file, of the type we described, all of the accounts would have to be rewritten on the output tape regardless of the activity ratio.

On a magnetic drum which is not constantly rewritten,[3] a certain space must be assigned to each record when the application has been established. Of course, on the basis of experience it might be changed from month to month, but we would still have this difficulty of the variation in the actual activity from the predicted activity either resulting in a low-storage efficiency or a high probability of overflow. In order to reduce this in a commercial checking-account application, the author investigated the possibility of a "combined record" technique.

If one were to investigate the degree of variability on an individual checking account over a long period of time, one would find that this variation was considerably greater than the variation in the combined activity for a large number of similar checking accounts over the same period of time. This is very familiar to statisticians, and others who have considered the implication of an averaging process.[4]

We might think of this averaging effect which reduces the variability of a group of accounts as follows: if 10 similar accounts were combined in a single record storage it would be expected that the variation in the space required for this kind of combined record would be less than the variation required for the 10 accounts if they were all kept separately, for the same degree of overflow. This result might be anticipated since the probability of member A of this group of ten being active at precisely the same time as member B is not very great except for such common activity increases as seasonal peaks. A given individual's activity for such personal transactions as buying a house, moving, purchasing an automobile, etc., would not be correlated with other individuals' activities of the same type. (We are not attempting the thesis that there is no correlation, but simply that correlation is considerably less than one.)

Specifically, if we wish to reduce the probability of overflow to some specified value, the amount of storage space required for the 10 combined accounts detail lumped into one storage area (*i.e.*, a record) would be considerably less than the amount of storage space required for the total of 10 individual accounts for the same probability of overflow. The author became quite interested in the possibilities of this technique and investigated in detail the special checking-account application at a large commercial bank.

Since it was considered too difficult to acquire a large amount of data over a long period on individual accounts, the approach was tried on a slightly different problem under the assumption that the same general conclusions would obtain in the case of the temporal variations as for the variations from account to account for a given time period. Hence, a significant sample of special checking accounts was examined for a given month. The distribution of the number of accounts vs the amount of activity in the account was obtained and is shown in Fig. 1.

[3] Incidentally, the same conclusions would apply to a magnetic-tape file in which the entire file was not rewritten but the new information inserted in the account-records storage on the tape.

[4] It can be proved that, regardless of what the individual distribution of activity might be over a period of time, if a sufficiently large number of them were to be combined, the distribution is Gaussian. The variance of the normal distribution would be less than that of the individual distributions.

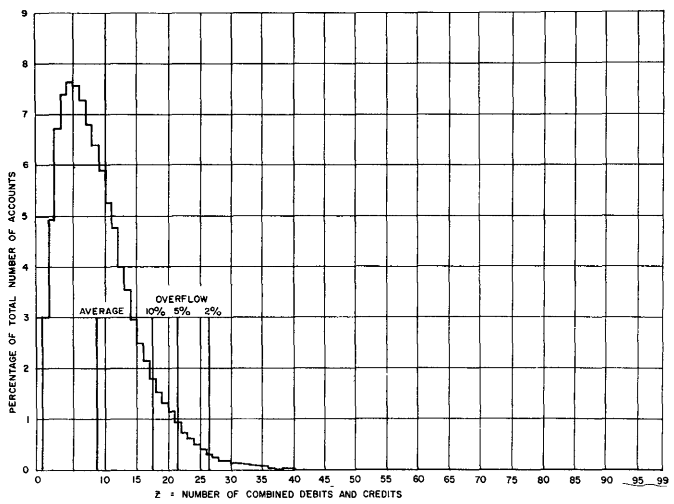


Fig. 1—Distribution of combined debits and credits. Special checking.

It is seen that, although the average activity for the special checking account was about 10 items, in order to reduce the probability of overflow to two per cent (have two per cent of the accounts overflowing), about 26 or 27 transactions would have to be accommodated. The samė results are shown in a cumulative distribution in Fig. 2. If, however, 16 (in this case 16 was selected because it is a power of two which made the calculations somewhat simpler) accounts were combined, the cumulative distribution shown in Fig. 3 results. Here, for a

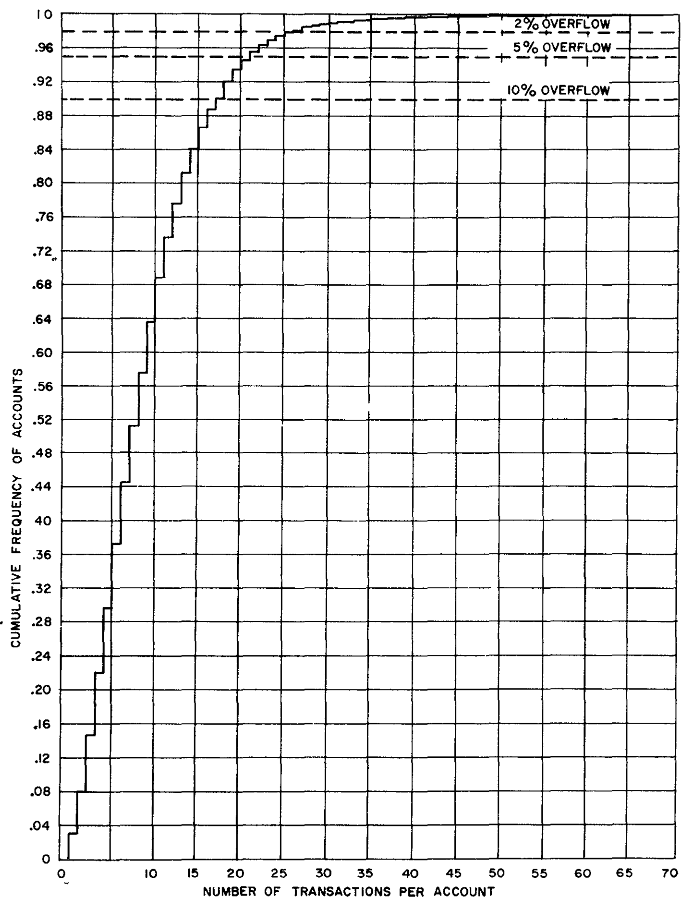


Fig. 2—Special checking.

two per cent overflow, the number of transactions to be stored could be reduced to 12. This resulted in a better than 2 to 1 reduction in storage space required.

The results are even more dramatic if one considers probabilities of overflow to be allowed to be considerably less. For example, if it were 0.1 per cent the number of transactions which must be provided for per account if the accounts are not grouped was 47. If 16 accounts were grouped, this number could be reduced to 14 per account thus resulting in a better than three to one reduction in storage requirements.

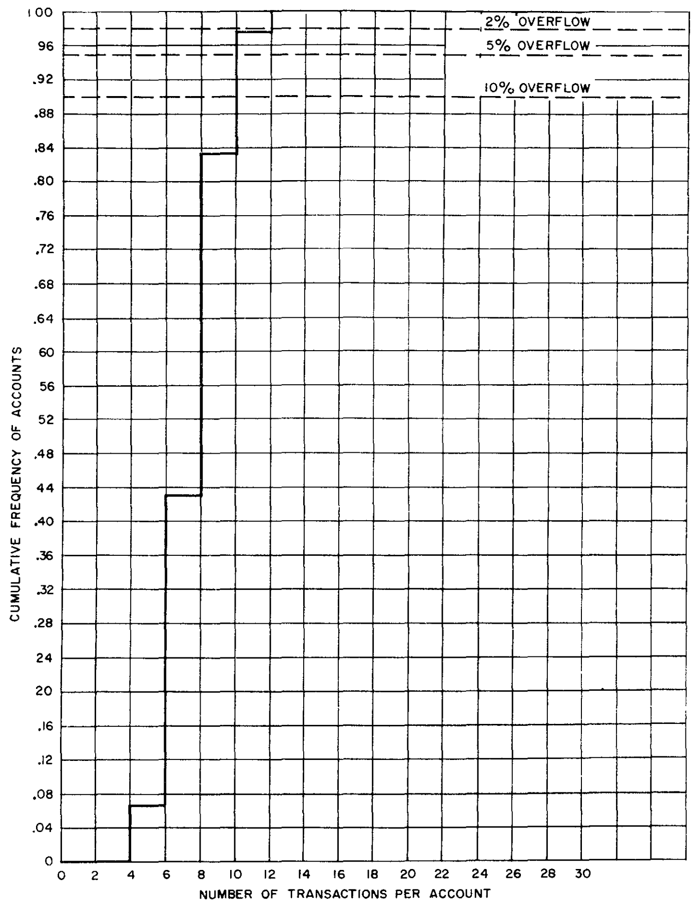


Fig. 3—Sixteen accounts grouped. Special checking.

If one were to plot the entire graph indicating the storage space required as a function of the number of accounts grouped for various probabilities of overflow, the results in Fig. 4 (next page) would be obtained.

Of course, it is obvious that there is a disadvantage to this system as opposed to having each individual account stored separately; *i.e.*, the fact that all of the transactions on the combined ten accounts have been lumped together. However, in at least one commercial data-processing system this disadvantage did not prove operationally serious owing to the ability to sort rapidly individual transactions from a group of transactions. The technique is based upon a single digit added to each item to indicate which one of the 10 accounts the item referred to (in the case where 10 transactions were grouped together).

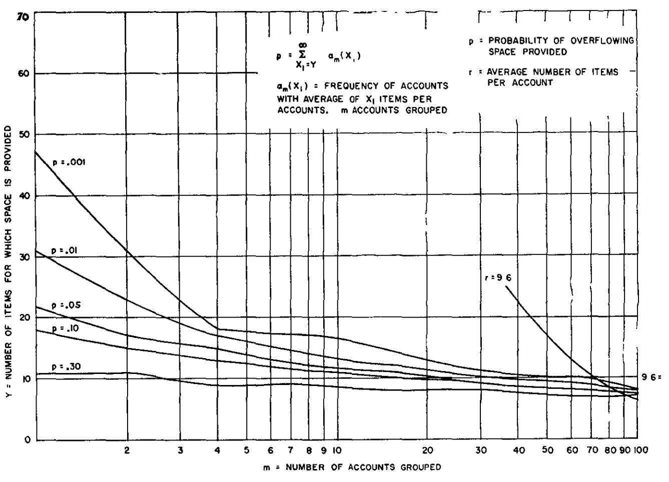


Fig. 4—Special checking accounts. Empirical distribution.

Looking at Fig. 4 we can see the significance of grouping 10 accounts if a one per cent overflow figure were to be tolerated. If the transactions had not been combined, a storage space of 31 transactions per account would have been required; but, with a combined-record approach, a storage space sufficient to accommodate a little over 13 items is adequate. Thus the storage space required is reduced by about 60 per cent. The technique for selecting which of the 10 accounts a given item belongs to on the basis of this single digit is beyond the scope of this paper and would depend upon the specific data processor utilized.

The author feels that this technique of combining similar accounts in a single record storage where the data processor can accommodate the sorting required is a very powerful one, indeed, in reducing the storage space required, particularly where storage space is at a premium as it is in magnetic-drum or core storage. As pointed out above, in many applications the advantages of this more expensive storage in the terms of more immediate random access are essential.

Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristic

A. NEWELL[†], J. C. SHAW[†], AND H. A. SIMON[‡]

THIS PAPER is a case study in problem solving, representing part of a program of research on complex information-processing systems. We have specified a system for finding proofs of theorems in elementary symbolic logic, and by programming a computer to these specifications, have obtained empirical data on the problem-solving process in elementary logic. The program is called the Logic Theory Machine (LT); it was devised to learn how it is possible to solve difficult problems such as proving mathematical theorems, discovering scientific laws from data, playing chess, or understanding the meaning of English prose.

The research reported here is aimed at understanding the complex processes (heuristics) that are effective in problem solving. Hence, we are not interested in methods that guarantee solutions, but which require vast amounts of computation. Rather, we wish to understand how a mathematician, for example, is able to prove a theorem even though he does not know when he starts how, or if, he is going to succeed.

This paper focusses on the pure theory of problem solving. In a previous paper[1] we specified in detail a program for the Logic Theory Machine; and we shall repeat here only as much of that specification as is needed so that the reader can understand our data. In a companion paper[2] we consider how computers can be programmed to execute processes of the kinds called for by LT, a problem that is interesting in its own right. Similarly, we postpone to later papers a discussion of the implications of our work for the psychological theory of human thinking and problem solving. Other areas of application will readily occur to the reader, but here we will limit our attention to the nature of the problem-solving process itself.

Our research strategy in studying complex systems is to specify them in detail, program them for digital computers, and study their behavior empirically by running them with a number of variations and under a variety of conditions. This appears at present the only adequate means to obtain a thorough understanding of their behavior. Although the problem area with which the present system, LT, deals is fairly elementary, it provides a good example of a difficult problem—logic is a subject taught in college courses, and is difficult enough for most humans.

Our data come from a series of programs run on the JOHNNIAC, one of RAND's high-speed digital computers. We will describe the results of these runs, and

† The RAND Corp., Santa Monica, Calif.
‡ Carnegie Institute of Technology, Pittsburgh, Pa.

[1] A. Newell and H. A. Simon, "The logic theory machine: a complex information processing system," IRE TRANS., vol. IT-2, pp. 61–79; September, 1956.

[2] A. Newell and J. C. Shaw, "Programming the logic theory machine," this issue, p. 230.

analyze and interpret their implications for the problem-solving process.

The Logic Theory Machine in Operation

We shall first give a concrete picture of the Logic Theory Machine in operation. LT, of course, is a program, written for the JOHNNIAC, represented by marks on paper or holes in cards. However, we can think of LT as an actual physical machine and the operation of the program as the behavior of the machine. One can identify LT with JOHNNIAC after the latter has been loaded with the basic program, but before the input of data.

LT's task is to prove theorems in elementary symbolic logic, or more precisely, in the sentential calculus. The sentential calculus is a formalized system of mathematics, consisting of expressions built from combinations of basic symbols. Five of these expressions are taken as axioms, and there are rules of inference for generating new theorems from the axioms and from other theorems. In flavor and form elementary symbolic logic is much like abstract algebra. Normally the variables of the system are interpreted as sentences, and the axioms and rules of inference as formalizations of logical operations, *e.g.*, deduction. However, LT deals with the system as a purely formal mathematics, and we will have no further need of the interpretation. We need to introduce a smattering of the sentential calculus to understand LT's task.

There is postulated a set of *variables* p, q, r, $\cdots$, A, B, C, $\cdots$, with which the sentential calculus deals. These variables can be combined into expressions by means of *connectives*. Given any variable p, we can form the expression "not-p." Given any two variables p and q, we can form the expression "p or q," or the expression "p implies q," where "or" and "implies" are the connectives. There are other connectives, for example "and," but we will not need them in this paper. Once we have formed expressions, these can be further combined into more complicated expressions. For example, we can form:[3]

$$\text{"}(p \text{ implies not-}p) \text{ implies not-}p.\text{"} \qquad (2.01)$$

There is also given a set of expressions that are axioms. These are taken to be the universally true expressions from which theorems are to be derived by means of various rules of inference. For the sake of definiteness in our work with LT, we have employed the system of axioms, definitions, and rules that is used in the "Principia Mathematica," which lists five axioms:

$$(p \text{ or } p) \text{ implies } p \qquad (1.2)$$
$$p \text{ implies } (q \text{ or } p) \qquad (1.3)$$
$$(p \text{ or } q) \text{ implies } (q \text{ or } p) \qquad (1.4)$$
$$[p \text{ or } (q \text{ or } r)] \text{ implies } [q \text{ or } (p \text{ or } r)] \qquad (1.5)$$
$$(p \text{ implies } q) \text{ implies } [(r \text{ or } p) \text{ implies } (r \text{ or } q)]. \qquad (1.6)$$

[3] For easy reference we have numbered axioms and theorems to correspond to their numbers in "Principia Mathematica," by A. N. Whitehead and B. Russell, Cambridge University Press, 2nd ed., vol. 1; 1935.

Given some true theorems one can derive new theorems by means of three rules of inference: *substitution*, *replacement*, and *detachment*.

1) By the rule of substitution, any expression may be substituted for any variable in any theorem, provided the substitution is made throughout the theorem wherever that variable appears. For example, by substitution of "p or q" for "p," in the second axiom we get the new theorem:

$$(p \text{ or } q) \text{ implies } [q \text{ or } (p \text{ or } q)].$$

2) By the rule of replacement, a connective can be replaced by its definition, and *vice versa*, in any of its occurrences. By definition "p implies q" means the same as "not-p or q." Hence the former expression can always be replaced by the latter and *vice versa*. For example from axiom (1.3), by replacing "implies" with "or," we get the new theorem:

$$\text{not-}p \text{ or } (q \text{ or } p).$$

3) By the rule of detachment, if "A" and "A implies B" are theorems, then "B" is a theorem. For example, from:

$$(p \text{ or } p) \text{ implies } p,$$

and

$$[(p \text{ or } p) \text{ implies } p] \text{ implies } (p \text{ implies } p),$$

we get the new theorem:

$$p \text{ implies } p.$$

Given an expression to prove, one starts from the set of axioms and theorems already proved, and applies the various rules successively until the desired expression is produced. The proof is the sequence of expressions, each one validly derived from the previous ones, that leads from the axioms and known theorems to the desired expression.

This is all the background in symbolic logic needed to observe LT in operation. LT "understands" expressions in symbolic logic—that is, there is a simple code for punching expressions on cards so they can be fed into the machine. We give LT the five axioms, instructing it that these are theorems it can assume to be true. LT already knows the rules of inference and the definitions—how to substitute, replace, and detach. Next we give LT a single expression, say, expression (2.01), and ask LT to find a proof for it. LT works for about 10 seconds and then prints out the following proof:

(p implies not-p) implies not-p	(theorem 2.01, to be proved)
1) (A or A) implies A	(axiom 1.2)
2) (not-A or not-A) implies not-A	(subs. of not-A for A)
3) (A implies not A) implies not-A	(repl. of "or" with "implies")
4) (p implies not-p) implies not-p	(subs. of p for A; *QED*).

Next we ask LT to prove a fairly advanced theorem,[4] theorem 2.45; allowing it to use all 38 theorems proved prior to 2.45. After about 12 minutes, LT produces the following proof:

[4] *Ibid.*, ch. 2.

not (p or q) implies not-p	(theorem 2.45, to be proved)
1) A implies (A or B)	(theorem 2.2)
2) p implies (p or q)	(subs. p for A, q for B in 1)
3) (A implies B) implies (not-B implies not-A)	(theorem 2.16)
4) [p implies (p or q)] implies [not (p or q) implies not-p]	[subs. p for A, (p or q) for B in 3]
5) not (p or q) implies not-p	(detach right side of 4, using 2; *QED*)

Finally, all the theorems prior to (2.31) are given to LT (a total of 28); and then LT is asked to prove:

$$[p \text{ or } (q \text{ or } r)] \text{ implies } [(p \text{ or } q) \text{ or } r]. \qquad (2.31)$$

LT works for about 23 minutes and then reports that it cannot prove (2.31), that it has exhausted its resources.

Now, what is there in this behavior of LT that needs to be explained? The specific examples given are difficult problems for most humans, and most humans do not know what processes they use to find proofs, if they find them. There is no known simple procedure that will produce such proofs. Various methods exist for verifying whether any given expression is true or false; the best known procedure is the method of truth tables. But these procedures do not produce a proof in the meaning of Whitehead and Russell. One can invent "automatic" procedures for producing proofs. We will look at one briefly later, but these turn out to require computing times of the orders of thousands of years for the proof of (2.45).

We must clarify why such problems are difficult in the first place, and then show what features of LT account for its successes and failures. These questions will occupy the rest of the paper.

Problems, Algorithms, and Heuristics

In describing LT, its environment, and its behavior we will make repeated use of three concepts. The first of these is the concept of *problem.* Abstractly, a person is given a problem if he is given a set of possible solutions, and a test for verifying whether a given element of this set is in fact a solution to his problem.

The reason why problems are problems is that the original set of possible solutions given to the problem solver can be very large, the actual solutions can be dispersed very widely and rarely throughout it, and the cost of obtaining each new element and of testing it can be very expensive. Thus the problem solver is not really "given" the set of possible solutions; instead he is given some process for generating the elements of that set in some order. This generator has properties of its own, not usually specified in stating the problem; *e.g.*, there is associated with it a certain cost per element produced, it may be possible to change the order in which it produces the elements, and so on. Likewise the verification test has costs and times associated with it. The problem can be solved if these costs are not too large in relation to the time and computing power available for solution.

One very special and valuable property that a generator of solutions sometimes has is a guarantee that if the problem has a solution, the generator will, sooner or later, produce it. We will call a process that has this property for some problem an *algorithm* for that problem. The guarantee provided by an algorithm is not an unmixed blessing, of course, since nothing has been specified about the cost or time required to produce the solutions. For example, a simple algorithm for opening a combination safe is to try all cominations, testing each one to see if it opens the safe. This algorithm is a typical problem-solving process: there is a generator that produces new combinations in some order, and there is a verifier that determines whether each new combination is in fact a solution to the problem. This search process is an algorithm because it is known that *some* combination will open the safe, and because the generator will exhaust all combinations in a finite interval of time. The algorithm is sufficiently expensive, however, that a combination safe can be used to protect valuables even from people who know the algorithm.

A process that *may* solve a given problem, but offers no guarantees of doing so, is called a *heuristic*[5] for that problem. This lack of a guarantee is not an unmixed evil. The cost inflicted by the lack of guarantee depends on what the process costs and what algorithms are available as alternatives. For most run-of-the-mill problems we have only heuristics, but occasionally we have both algorithms and heuristics as alternatives for solving the same problem. Sometimes, as in the problem of finding maxima for simple differentiable functions, everyone uses the algorithm of setting the first derivative equal to zero; no one sets out to examine all the points on the line one by one even if it were possible. Sometimes, as in chess, everyone plays by heuristic, since no one is able to carry out the algorithm of examining all continuations of the game to termination.

The Problem of Proving Theorems in Logic

Finding a proof for a theorem in symbolic logic can be described as selecting an element from a generated set, as shown by Fig. 1. Consider the *set of all possible sequences of logic expressions*—call it E. Certain of these sequences, a very small minority, will be proofs. A proof sequence satisfies the following test:

Each expression in the sequence is either

1) One of the accepted theorems or axioms, or
2) Obtainable from one or two previous expressions in the sequence by application of one of the three rules of inference.

[5] As a noun, "heuristic" is rare and generally means the art of discovery. The adjective "heuristic" is defined by Webster as: serving to discover or find out. It is in this sense that it is used in the phrase "heuristic process" or "heuristic method." For conciseness, we will use "heuristic" in this paper as a noun synonymous with "heuristic process." No other English word appears to have this meaning.

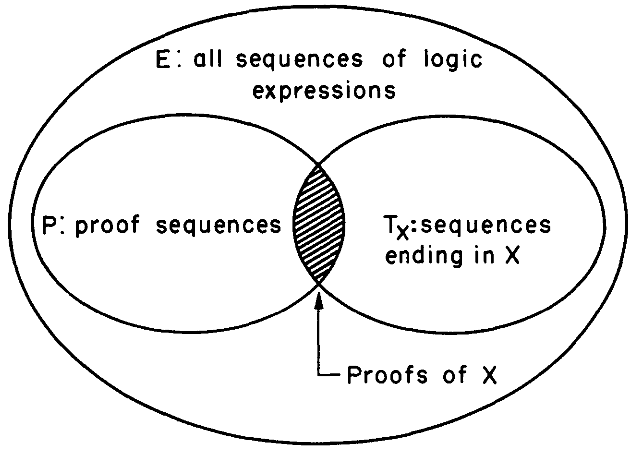

E: all sequences of logic expressions
P: proof sequences
T_X:sequences ending in X
Proofs of X

Fig. 1—Relationships between E, P, and T_X.

Call the *set of sequences that are proofs P*. Certain of the sequences in E have the *expression to be proved*—call it X, as their final expression. Call this set of sequences T_X. Then, to find a proof of a given theorem X means to select an element of E that belongs to the intersection of P and T_X. The set E is given implicitly by rules for generating new sequences of logic expressions.

The difficulty of proving theorems depends on the scarcity of elements in the intersection of P and T_X, relative to the number of elements in E. Hence, it depends on the cost and speed of the available generators that produce elements of E, and on the cost and speed of making tests that determine whether an element belongs to T_X or P. The difficulty also depends on whether generators can be found that guarantee that any element they produce automatically satisfies some of the conditions. Finally, as we shall see, the difficulty depends heavily on what heuristics can be found to guide the selection.

A little reflection, and experience in trying to prove theorems, make it clear that proof sequences for specified theorems are rare indeed. To reveal more precisely why proving theorems is difficult, we will construct an algorithm for doing this. The algorithm will be based only on the tests and definitions given above, and not on any "deep" inferred properties of symbolic logic. Thus it will reflect the basic nature of theorem proving; that is, its nature prior to building up sophisticated proof techniques. We will call this algorithm the British-Museum algorithm, in recognition of the supposed originators of procedures of this type.

The British-Museum Algorithm

The algorithm constructs all possible proofs in a systematic manner, checking each time 1) to eliminate duplicates, and 2) to see if the final theorem in the proof coincides with the expression to be proved. With this algorithm the set of one-step proofs is identical with the set of axioms (*i.e.*, each axiom is a one-step proof of itself). The set of n-step proofs is obtained from the set of $(n-1)$-step proofs by making all the permissible

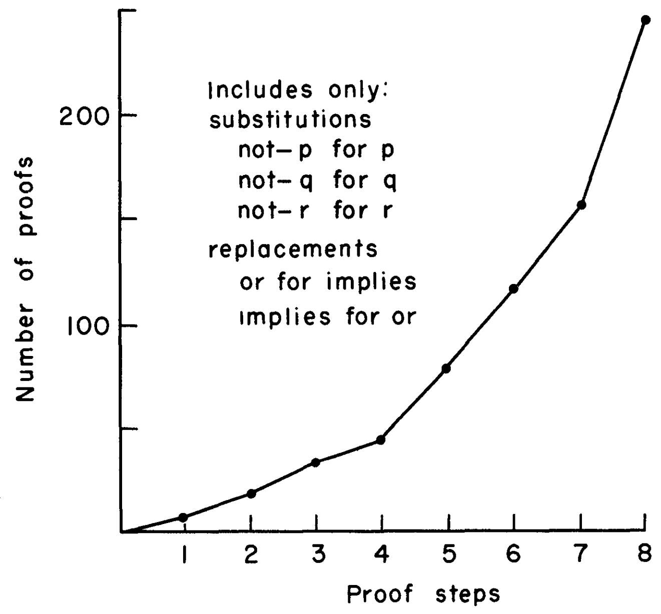


Fig. 2—Number of proofs generated by first few steps of British Museum algorithm.

substitutions and replacements in the expressions of the $(n-1)$-step proofs, and by making all the permissible detachments of pairs of expressions as permitted by the recursive definition of proof.[6]

Fig. 2 shows how the set of n-step proofs increases with n at the very start of the proof-generating process. This enumeration only extends to replacements of "or" with "implies," "implies" with "or," and negation of variables (*e.g.*, "not-p" for "p"). No detachments and no complex substitutions (*e.g.*, "q or r" for "p") are included. No specializations have been made (*e.g.*, subsitution of p for q in "p or q"). If we include the specializations, which take three more steps, the algorithm will generate an (estimated) additional 600 theorems, thus providing a set of proofs of 11 steps or less containing almost 1000 theorems, none of them duplicates.

In order to see how this algorithm would provide proofs of specified theorems, we can consider its performance on the sixty-odd theorems of Chapter 2 of "Principia." One theorem (2.01) is obtained in step (4) of the generation, hence is among the first 42 theorems proved. Three more (2.02, 2.03, and 2.04) are obtained in step (6), hence among the first 115. One more (2.05) is obtained in step (8), hence in the first 246. Only one more is included in the first 1000, theorem 2.07. The proofs of all the remainder require complex substitutions or detachment.

We have no way at present to estimate how many proofs must be generated to include proofs of all theorems of Chapter 2 of "Principia." Our best guess is that it

[6] A number of fussy but not fundamental points must be taken care of in constructing the algorithm. The phrase "all permissible substitutions" needs to be qualified, for there is an infinity of these. Care must be taken not to duplicate expressions that differ only in the names of their variables. We will not go into details here, but simply state that these difficulties can be removed. The essential feature in constructing the algorithm is to allow only one thing to happen in generating each new expression, *i.e.*, one replacement, substitution of "not-p" for "p," etc.

might be a hundred million. Moreover, apart from the six theorems listed, there is no reason to suppose that the proofs of these theorems would occur early in the list.

Our information is too poor to estimate more than very roughly the times required to produce such proofs by the algorithm; but we can estimate times of about 16 minutes to do the first 250 theorems of Fig. 2 [*i.e.*, through step (8)] assuming processing times comparable with those in LT. The first part of the algorithm has an additional special property, which holds only to the point where detachment is first used; that no check for duplication is necessary. Thus the time of computing the first few thousand proofs only increases linearly with the number of theorems generated. For the theorems requiring detachments, duplication checks must be made, and the total computing time increases as the square of the number of expressions generated. At this rate it would take hundreds of thousands of years of computation to generate proofs for the theorems in Chapter 2.

The nature of the problem of proving theorems is now reasonably clear. When sequences of expressions are produced by a simple and cheap (per element produced) generator, the chance that any particular sequence is the desired proof is exceedingly small. This is true even if the generator produces sequences that always satisfy the most complicated and restrictive of the solution conditions: that each is a proof of something. The set of sequences is so large, and the desired proof so rare, that no practical amount of computation suffices to find proofs by means of such an algorithm.

The Logic Theory Machine

If LT is to prove any theorems at all it must employ some devices that alter radically the order in which possible proofs are generated, and the way in which they are tested. To accomplish this, LT gives up almost all the guarantees enjoyed by the British-Museum algorithm. Its procedures guarantee neither that its proposed sequences are proofs of something, nor that LT will ever find the proof, no matter how much effort is spent. However, they *often* generate the desired proof in a reasonable computing time.

Methods

The major type of heuristic that LT uses we call a *method.* As yet we have no precise definition of a method that distinguishes it from all the other types of routines in LT. Roughly, a method is a reasonably self-contained operation that, if it works, makes a major and permanent contribution toward finding a proof. It is the largest unit of organization in LT, subordinated only to the executive routines necessary to coordinate and select the methods.

The Substitution Method: This method seeks a proof for the problem expression by finding an axiom or previously proved theorem that can be transformed, by a series of substitutions for variables and replacements of connectives, into the problem expression.

The Detachment Method: This method attempts, using the rule of detachment, to substitute for the problem expression a new subproblem which, if solved, will provide a proof for the problem expression. Thus, if the problem expression is B, the method of detachment searches for an axiom or theorem of the form "A implies B." If one is found, A is set up as a new subproblem. If A can be proved, then, since "A implies B" is a theorem, B will also be proved.

The Chaining Methods: These methods use the transitivity of the relation of implication to create a new subproblem which, if solved, will provide a proof for the problem expression. Thus, if the problem expression is "a implies c," the method of forward chaining searches for an axiom or theorem of the form "a implies b." If one is found, "b implies c" is set up as a new subproblem. Chaining backward works analogously: it seeks a theorem of the form "b implies c," and if one is found, "a implies b" is set up as a new subproblem.

Each of these methods is an independent unit. They are alternatives to one another, and can be used in sequence, one working on the subproblems generated by another. Each of them produces a major part of a proof. Substitution actually proves theorems, and the other three generate subproblems, which can become the intermediate expressions in a proof sequence.

These methods give no guarantee that they will work. There is no guarantee that a theorem can be found that can be used to carry out a proof by the substitution method, or a theorem that will produce a subproblem by any of the other three methods. Even if a subproblem is generated, there is no guarantee that it is part of the desired proof sequence, or even that it is part of any proof sequence (*e.g.*, it can be false). On the other hand, the generated methods do guarantee that any subproblem generated is part of a sequence of expressions that ends in the desired theorem (this is one of the conditions that a sequence be a proof). The methods also guarantee that each expression of the sequence is derived by the rules of inference from the preceding ones (a second condition of proof). What is not guaranteed is that the beginning of the sequence can be completed with axioms or previously proved theorems.

There is also no guarantee that the combination of the four methods, used in any fashion whatsoever and with unlimited computing effort, comprises a sufficient set of methods to prove all theorems. In fact, we have discovered a theorem [(2.13), "p or not-not-not-p"] which the four methods of LT cannot prove. All the subproblems generated for (2.13) after a certain point are false, and therefore cannot lead to a proof.

We have yet no general theory to explain why the methods transform LT into an effective problem solver. That they do, in conjunction with the other mechanisms to be described shortly, will be demonstrated amply in the remainder of the paper. Several factors may be involved. First, the methods organize the sequences of individual processing steps into larger units that can be handled as such. Each processing step can be oriented toward the special function it performs in the unit as a whole, and the units can be manipulated and organized as entities by the higher-level routines.

Apart from their "unitizing" effect, the methods that generate subproblems work "backwards" from the desired theorem to axioms or known theorems rather than "forward" as did the British-Museum algorithm. Since there is only one theorem to be proved, but a number of known true theorems, the efficacy of working backward may be analogous to the ease with which a needle can find its way out of a haystack, compared with the difficulty of someone finding the lone needle in the haystack.

The Executive Routine

In LT the four methods are organized by an executive routine, whose flow diagram is shown in Fig. 3.

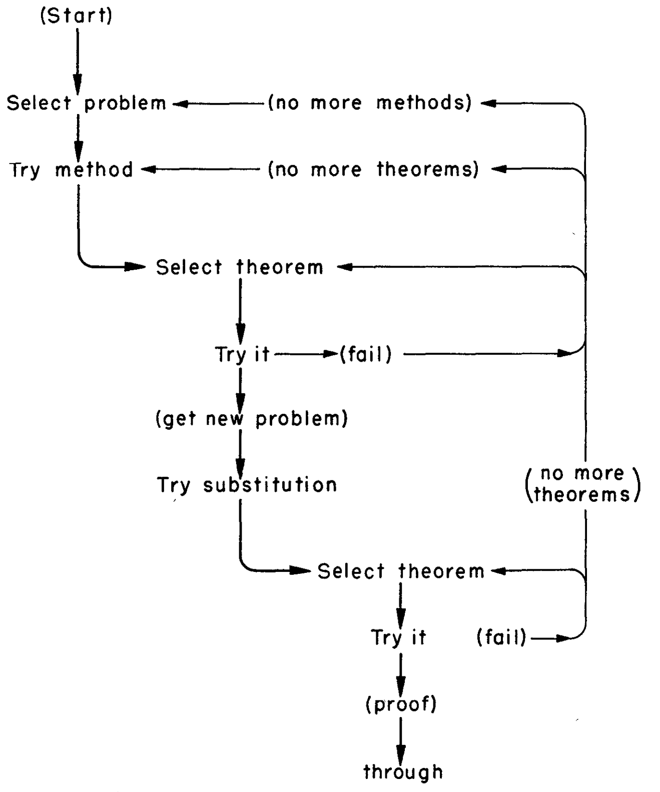


Fig. 3—General flow diagram of LT.

1) When a new problem is presented to LT, the substitution method is tried first, using all the axioms and theorems that LT has been told to assume, and that are now stored in a *theorem list.*

2) If substitution fails, the detachment method is tried, and as each new subproblem is created by a successful detachment, an attempt is made to prove the new subproblem by the substitution method. If substitution fails again, the subproblem is added to a *subproblem list.*

3) If detachment fails for all the theorems in the theorem list, the same cycle is repeated with forward chaining, and then with backward chaining: try to create a subproblem; try to prove it by the substitution method; if unsuccessful, put the new subproblem on the list. By the nature of the methods, if the substitution method ever succeeds with a single subproblem, the original theorem is proved.

4) If all the methods have been tried on the original problem and no proof has been produced, the executive routine selects the next untried subproblem from the subproblem list, and makes the same sequence of attempts with it. This process continues until 1) a proof is found, 2) the time allotted for finding a proof is used up, 3) there is no more available memory space in the machine, or 4) no untried problems remain on the subproblem list.

In the three examples cited earlier, the proof of (2.01) [(p implies not-p) implies not-p] was obtained by the substitution method directly, hence did not involve use of the subproblem list.

The proof of (2.45) [not (p or q) implies not-p] was achieved by an application of the detachment method followed by a substitution. This proof required LT to create a subproblem, and to use the substitution method on it. It did not require LT ever to select any subproblem from the subproblem list, since the substitution was successful. Fig. 4 shows the *tree of subproblems*

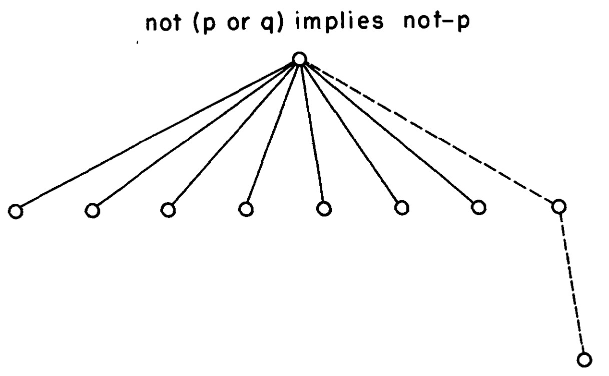


Fig. 4—Subproblem tree of proof by LT of 2.45 (all previous theorems available).

corresponding to the proof of (2.45). The subproblems are given in the form of a downward branching tree. Each node is a subproblem, the original problem being the single node at the top. The lines radiating down from a node lead to the new subproblems generated from the subproblem corresponding to the node. The proof sequence is given by the dashed line; the top link was constructed by the detachment method, and the bottom link by the substitution method. The other links extending down from the original problem lead to other subproblems generated by the detachment method (but not provable by direct substitution) prior to the time LT tried the theorem that leads to the final proof.

LT did not prove theorem 2.31, also mentioned earlier, and gave as its reason that it could think of nothing more to do. This means that LT had considered all subproblems on the subproblem list (there were six in this case) and had no new subproblems to work on. In none of the examples mentioned did LT terminate because of time or space limitations; however, this is the most common result in the cases where LT does not find a proof. Only rarely does LT run out of things to do.

This section has described the organization of LT in terms of methods. We have still to examine in detail why it is that this organization, in connection with the additional mechanisms to be described below, allows LT to prove theorems with a reasonable amount of computing effort.

The Matching Process

The times required to generate proofs for even the simplest theorems by the British-Museum algorithm are larger than the times required by LT by factors ranging from five (for one particular theorem) to a hundred and upwards. Let us consider an example from the earliest part of the generation, where we have detailed information about the algorithm. The 79th theorem generated by the algorithm (see Fig. 2) is theorem 2.02 of "Principia," one of the theorems we asked LT to prove. This theorem, "p implies (q implies p)," is generated by the algorithm in about 158 seconds with a sequence of substitutions and replacements; it is proved by LT in about 10 seconds with the method of substitution. The reason for the difference becomes apparent if we focus attention on axiom 1.3, "p implies (q or p)," from which the theorem is derived in either scheme.

Fig. 5 shows the tree of proofs of the first twelve theorems obtained from (1.3) by the algorithm. The theorem 2.02 is node (9) on the tree and is obtained by substitution of "not-q" for "q" in axiom 1.3 to reach node (5); and then by replacing the "(not-q or p)" by "(q implies p)" in (5) to get (9). The 9th theorem generated from axiom 1.3 is the 79th generated from the five axioms considered together.

This proof is obtained directly by LT using the following *matching* procedure. We compare the axiom with (9), the expression to be proved:

$$p \text{ implies } (q \text{ or } p) \qquad (1.3)$$
$$p \text{ implies } (q \text{ implies } p). \qquad (9)$$

First, by a direct comparison, LT determines that the main connectives are identical. Second, LT determines that the variables to the left of the main connectives are identical. Third, LT determines that the connectives within parentheses on the right-hand sides are different. It is necessary to replace the "or" with "implies," but in order to do this (in accordance with the definition of implies) there must be a negation sign before the variable that precedes the "or." Hence, LT first replaces the "q" on the right-hand side with "not-q" to get the required negation sign, obtaining (5). Now LT can change the "or" to "implies," and determines that the resulting expression is identical with (9).

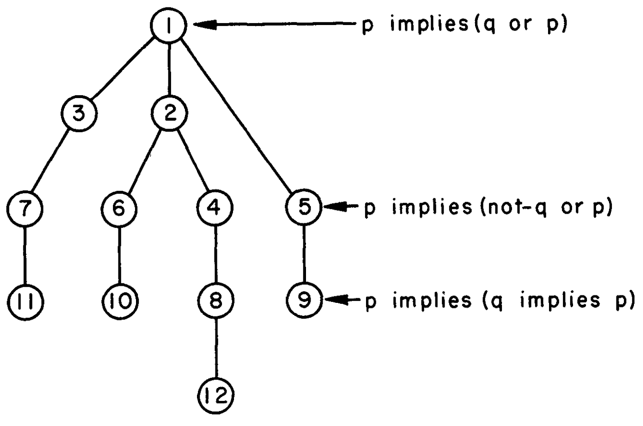


Fig. 5—Proof tree of proof of 2.02 by British Museum algorithm (using axiom 1.3).

The matching process allowed LT to proceed directly down the branch from (1) through (5) to (9) without even exploring the other branches. Quantitatively, it looked at only two expressions instead of eight, thus reducing the work of comparison by a factor of four. Actually, the saving is even greater, since the matching procedure does not deal with whole expressions, but with a single pair of elements at a time.

An important source of efficiency in the matching process is that it proceeds component-wise, obtaining at each step a feedback of the results of a substitution or replacement that can be used to guide the next step. This feedback keeps the search on the right branch of the tree of possible expressions. It is not important for an efficient search that the goal be known from the beginning; it is crucial that hints of "warmer" or "colder" occur as the search proceeds.[7] Closely related to this feedback is the fact that where LT is called on to make a substitution or replacement at any step, it can determine immediately what variable or connective to substitute or replace by direct comparison with the problem expression, and without search.

Thus far we have assumed that LT knows at the beginning that (1.3) is the appropriate axiom to use. Without this information, it would begin matching with each axiom in turn, abandoning it for the next one if the matching should prove impossible. For example, if it tries to match the theorem against axiom 1.2, it determines almost immediately (on the second test) that "p or p" cannot be made into "p" by substitution. Thus, the matching process permits LT to abandon unprofitable lines of search as well as guiding it to correct substitutions and replacements.

[7] The following analogy may be instructive. Changing the symbols in a logic expression until the "right" expression is obtained is like turning the dials on a safe until the right combination is obtained. Suppose two safes, each with ten dials and ten numbers on a dial. The first safe gives a signal (a "click") when any given dial is turned to the correct number; the second safe clicks only when all ten dials are correct. Trial-and-error search will open the first safe, on the average, in 50 trials; the second safe, in five billion trials.

Matching in the Substitution Method: The matching process is an essential part of the substitution method. Without it, the substitution method is just that part of the British-Museum algorithm that uses only replacements and substitutions. With it, LT is able, either directly or in combination with the other methods, to prove many theorems with reasonable effort.

To obtain data on its performance, LT was given the task of proving in sequence the first 52 theorems of "Principia." In each case, LT was given the axioms plus all the theorems previously proved in Chapter 2 as the material from which to work (regardless of whether LT had proved the theorems itself).[8]

Of the 52 theorems, proofs were found for a total 38 (73 per cent). These proofs were obtained by various combinations of methods, but the substitution method was an essential component of all of them. Seventeen of these proofs, almost a half, were accomplished by the substitution method alone. Subjectively evaluated, the theorems that were proved by the substitution method alone have the appearance of "corollaries" of the theorems they are derived from; they occur fairly close to them in the chapter, generally requiring three or fewer attempts at matching per theorem proved (54 attempts for 17 theorems).

The performance of the substitution method on the subproblems is somewhat different, due, we think, to the kind of selectivity implicit in the order of theorems in "Principia." In 338 attempts at solving subproblems by substitution, there were 21 successes (6.2 per cent). Thus, there was about one chance in three of proving an original problem directly by the substitution method, but only about one chance in 16 of so proving a subproblem generated from the original problem.

Matching in Detachment and Chaining: So far the matching process has been considered only as a part of the substitution method, but it is also an essential component of the other three methods. In detachment, for example, a theorem of form "A implies B" is sought, where B is identical with the expression to be proved. The chances of finding such a theorem are negligible unless we allow some modification of B to make it match the theorem to be proved. Hence, once a theorem is selected from the theorem list, its right-hand subexpression is matched against the expression to be proved. An analogous procedure is used in the chaining methods.

We can evaluate the performance of the detachment and chaining methods with the same sample of problems used for evaluating the substitution method. However, a successful match with the former three methods generates a subproblem and does not directly prove the theorem. With the detachment method, an average of three new subproblems were generated for each application of the method; with forward chaining the average was 2.7; and with backward chaining the average was 2.2. For all the methods, this represents about one subproblem per $7\frac{1}{2}$ theorems tested (the number of theorems available varied slightly).

As in the case of substitution, when these three methods were applied to the original problem, the chances of success were higher than when they were applied to subproblems. When applied to the original problem, the number of subproblems generated averaged eight to nine; when applied to subproblems derived from the original, the number of subproblems generated fell to an average of two or three.

In handling the first 52 problems in Chapter 2 of "Principia," 17 theorems were proved in one step—that is, in one application of substitution. Nineteen theorems were proved in two steps, 12 by detachment followed by substitution, and seven by chaining forward followed by substitution. Two others were proved in three steps. Hence, 38 theorems were proved in all. There are no two step proofs by backward chaining, since, for two step proofs only, if there is a proof by backward chaining, there is also one by forward chaining. In 14 cases LT failed to find a proof. Most of these unsuccessful attempts were terminated by time or space limitations. One of these 14 theorems we know LT cannot prove, and one other we believe it cannot prove. Of the remaining twelve, most of them can be proved by LT if it has sufficient time and memory (see section on subproblems, however).

Similarity Tests and Descriptions

Matching eliminates enough of the trial and error in substitutions and replacements to make LT into a successful problem solver. Matching permeates all of the methods, and without it none of them would be useful within practical amounts of computing effort. However, a large amount of search is still used in finding the correct theorems with which matching works. Returning to the performance of LT in Chapter 2, we find that the over-all chances of a particular match being successful are 0.3 per cent for substitution, 13.4 per cent for detachment, 13.8 per cent for forward chaining, and 9.4 per cent for backward chaining.

The amount of search through the theorem list can be reduced by interposing a screening process that will reject any theorem for matching that has low likelihood of success. LT has such a screening device, called the *similarity test.* Two logic expressions are defined to be similar if both their left-hand and right-hand sides are equal, with respect to, 1) the maximum number of *levels* from the main connective to any variable; 2) the number of *distinct* variables; and 3) the number of *variable places.* Speaking intuitively, two logic expressions are "similar" if they look alike, and look alike if they are similar. Consider for example:

[8] The version of LT used for seeking solutions of the 52 problems included a similarity test (see next section). Since the matching process is more important than the similarity test, we have presented the facts about matching first, using adjusted statistics. A notion of the sample sizes can be gained from Table I. The sample was limited to the first 52 of the 67 theorems in Chapter 2 of "Principia" because of memory limitations of JOHNNIAC.

$$(p \text{ or } q) \text{ implies } (q \text{ or } p) \quad (1)$$
$$p \text{ implies } (q \text{ or } p) \quad (2)$$
$$r \text{ implies } (m \text{ implies } r). \quad (3)$$

By the definition of similarity, (2) and (3) are similar, but (1) is not similar to either (2) or (3).

In all of the methods LT applies the similarity tests to all expressions to be matched, and only applies the matching routine if the expressions are similar; otherwise it passes on to the next theorem in the theorem list. The similarity test reduces substantially the number of matchings attempted, as the numbers in Table I show, and correspondingly raises the probability of a match if the matching is attempted. The effect is particularly strong in substitution, where the similarity test reduces the matchings attempted by a factor of ten, and increases the probability of a successful match by a factor of ten. For the other methods attempted matchings were reduced by a factor of four or five, and the probability of a match increased by the same factor.

TABLE I
STATISTICS OF SIMILARITY TESTS AND MATCHING

Method	Theorems Considered	Theorems Similar	Theorems Matched	Per Cent Similar of Theorems Considered	Per Cent Matched of Theorems Similar
Substitution	11,298	993	37	8.8	3.7
Detachment	1,591	406	210	25.5	51.7
Chain. Forward	869	200	120	23.0	60.0
Chain. Backward	673	146	63	21.7	43.2

These figures reveal a gross, but not necessarily a net, gain in performance through the use of the similarity test. There are two reasons why all the gross gain may not be realized. First, the similarity test is only a heuristic. It offers no guarantee that it will let through only expressions that will subsequently match. The similarity test also offers no guarantee that it will not reject expressions that would match if attempted. The similarity test does not often commit this type of error (corresponding to a type II statistical error), as will be shown later. However, even rare occurrences of such errors can be costly. One example occurs in the proof of theorem 2.07:

$$p \text{ implies } (p \text{ or } p). \quad (2.07)$$

This theorem is proved simply by substituting p for q in axiom 1.3:

$$p \text{ implies } (q \text{ or } p). \quad (1.3)$$

However, the similarity test, because it demands equality in the number of distinct variables on the right-hand side, calls (2.07) and (1.3) dissimilar because (2.07) contains only p while (1.3) contains p and q. LT discovers the proof through chaining forward, where it checks for a direct match before creating the new subproblem, but the proof is about five times as expensive as when the similarity test is omitted.

The second reason why the gross gain will not all be realized is that the similarity test is not costless, and in fact for those theorems which pass the test the cost of the similarity test must be paid in addition to the cost of the matching. We will examine these costs in the next section when we consider the effort LT expends.

Experiments have been carried out with a weaker similarity test, which compares only the number of variable places on both sides of the expression. This test will not commit the particular type II error cited above, and (2.07) is proved by substitution using it. Apart from this, the modification had remarkably little effect on performance. On a sample of ten problems it admitted only 10 per cent more similar theorems and about 10 per cent more subproblems. The reason why the two tests do not differ more radically is that there is a high correlation among the descriptive measures.

Effort in LT

So far we have focussed entirely on the performance characteristics of the heuristics in LT, except to point out the tremendous difference between the computing effort required by LT and by the British-Museum algorithm. However, it is clear that each additional test, search, description, and the like, has its costs in computing effort as well as its gains in performance. The costs must always be balanced against the performance gains, since there are always alternative heuristics which could be added to the system in place of those being used. In this section we will analyze the computing effort used by LT. The memory space used by the various processes also constitutes a cost, but one that will not be discussed in this paper.

Measuring Effort: LT is written in an interpretive language or pseudo code, which is described in the companion paper to this one. LT is defined in terms of a set of primitive operations, which, in turn, are defined by subroutines in JOHNNIAC machine language. These primitives provide a convenient unit of effort, and all effort measurements will be given in terms of total number of primitives executed. The relative frequencies of the different primitives are reasonably constant, and, therefore, the total number of primitives is an adequate index of effort. The average time per primitive is quite constant at about 30 milliseconds, although for very low totals (less than 1000 primitives) a figure of about 20 milliseconds seems better.

Computing Effort and Performance: On *a priori* grounds we would expect the amount of computing effort required to solve a logic problem to be roughly proportional to the total number of theorems examined (*i.e.*, tested for similarity, if there is a similarity routine; or tested for matching, if there is not) by the various methods in the course of solving the problem. In fact, this turns out to be a reasonably good predictor of effort; but the fit to data is much improved if we assign greater weight to theorems considered for detachment and chaining than to theorems considered for substitution.

Actual and predicted efforts are compared below (with the full similarity test included, and excluding theorems proved by substitution) on the assumption that the number of primitives per theorem considered is twice as great for chaining as for substitution, and three times as great for detachment. About 45 primitives are executed per theorem considered with the substitution method (hence 135 with detachment and 90 with chaining). As Table II shows, the estimates are generally accurate within a few per cent, except for theorem 2.06, for which the estimate is too low.

TABLE II
EFFORT STATISTICS WITH "PRECOMPUTE DESCRIPTION" ROUTINE

Total Primitives (in thousands)		
Theorem	Actual	Estimate
2.06	3.2	0.8
2.07	4.3	4.4
2.08	3.5	3.3
2.11	2.2	2.2
2.13	24.5	24.6
2.14	3.3	3.2
2.15	15.8	13.6
2.18	34.1	35.8
2.25	11.1	11.5

There is an additional source of variation not shown in the theorems selected for Table II. The descriptions used in the similarity test must be computed from the logic expressions. Since the descriptions of the theorems are used over and over again, LT computes these at the start of a problem and stores the values with the theorems, so they do not have to be computed again. However, as the number of theorems increases, the space devoted to storing the precomputed descriptions becomes prohibitive, and LT switches to recomputing them each time it needs them. With recomputation, the problem effort is still roughly proportional to the total number of theorems considered, but now the number of primitives per theorem is around 70 for the substitution method, 210 for detachment, and 140 for chaining.

Our analysis of the effort statistics shows, then, that in the first approximation the effort required to prove a theorem is proportional to the number of theorems that have to be considered before a proof is found; the number of theorems considered is an effort measure for evaluating a heuristic. A good heuristic, by securing the consideration of the "right" theorems early in the proof, reduces the expected number of theorems to be considered before a proof is found.

Evaluation of the Similarity Test: As we noted in the previous section, to evaluate an improved heuristic, account must be taken of any additional computation that the improvement introduces. The net advantage may be less than the gross advantage, or the extra computing effort may actually cancel out the gross gain in selectivity. We are now in a position to evaluate the similarity routines as preselectors of theorems for matching.

A number of theorems were run, first with the full similarity routine, then with the modified similarity routine (which tests only the number of variable places), and finally with no similarity test at all. We also made some comparisons with both precomputed and recomputed descriptions.

When descriptions are precomputed, the computing effort is less with the full similarity test than without it; the factor of saving ranged from 10 to 60 per cent (*e.g.*, 3534/5206 for theorem 2.08). However, if LT must recompute the descriptions every time, the full similarity test is actually more expensive than no similarity test at all (*e.g.*, 26,739/22,914 for theorem 2.45).

The modified similarity test fares somewhat better. For example, in proving (2.45) it requires only 18,035 primitives compared to the 22,914 for no similarity test (see the paragraph above). These comparisons involve recomputed descriptions; we have no figures for precomputed descriptions, but the additional saving appears small since there is much less to compute with the abridged than with the full test.

Thus the similarity test is rather marginal, and does not provide anything like the factors of improvement achieved by the matching process, although we have seen that the performance figures seem to indicate much more substantial gains. The reason for the discrepancy is not difficult to find. In a sense, the matching process consists of two parts. One is a testing part that locates the differences between elements and diagnoses the corrective action to be taken. The other part comprises the processes of substituting and replacing. The latter part is the major expense in a matching that works, but most of this effort is saved when the matching fails. Thus matching turns out to be inexpensive for precisely those expressions that the similarity test excludes.

SUBPROBLEMS

LT can prove a great many theorems in symbolic logic. However, there are numerous theorems that LT cannot prove, and we may describe LT as having reached a plateau in its problem solving ability.

Fig. 6, (next page) shows the amount of effort required for the problems LT solved out of the sample of 52. Almost all the proofs that LT found took less than 30,000 primitives of effort. Among the numerous attempts at proofs that went beyond this effort limit, only a few succeeded, and these required a total effort that was very much greater.

The predominance of short proofs is even more striking than the approximate upper limit of 30,000 primitives suggests. The proofs by substitution—almost half of the total—required about 1000 primitives or less each. The effort required for the longest proof—89,000 primitives—is some 250 times the effort required for the short proofs. We estimate that to prove the 12 additional theorems that we believe LT can prove requires the effort limit to be extended to about a million primitives.

From these data we infer that LT's power as a problem solver is largely restricted to problems of a certain class. While it is logically possible for LT to solve others by large expenditures of effort, major adjustments are needed in the program to extend LT's powers to essen-

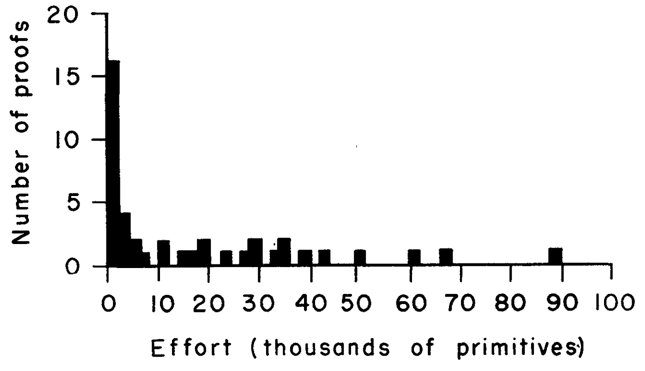


Fig. 6—Distribution of LT's proofs by effort. Data include all proofs from attempts on the first 52 theorems in Chapter 2 of "Principia."

tially new classes of problems. We believe that this situation is typical: good heuristics produce differences in performance of large orders of magnitude, but invariably a "plateau" is reached that can be surpassed only with quite different heuristics. These new heuristics will again make differences of orders of magnitude. In this section we shall analyze LT's difficulties with those theorems it cannot prove, with a view to indicating the general type of heuristic that might extend its range of effectiveness.

The Subproblem Tree

Let us examine the proof of theorem 2.17 when all the preceding theorems are available. This is the proof that cost LT 89,000 primitives. It is reproduced below, using chaining as a rule of inference (each chaining could be expanded into two detachments, to conform strictly to the system of "Principia").

(not-*q* implies not-*p*) implies (*p* implies *q*)	(theorem 2.17, to be proved)
1) *A* implies not-not-*A*	(theorem 2.12)
2) *p* implies not-not-*p*	(subs. *p* for *A* in 1)
3) (*A* implies *B*) implies [(*B* implies *C*) implies (*A* implies *C*)]	(theorem 2.06)
4) (*p* implies not-not-*p*) implies [(not-not-*p* implies *q*) implies (*p* implies *q*)]	(subs. *p* for *A*, not-not-*p* for *B*, *q* for *C* in 3)
5) (not-not-*p* implies *q*) implies (*p* implies *q*)	(det. 4 from 3)
6) (not-*A* implies *B*) implies (not-*B* implies *A*)	(theorem 2.15)
7) (not-*q* implies not-*p*) implies (not-not-*p* implies *q*)	(subs. *q* for *A*, not-*p* for *B*)
8) (not-*q* implies not-*p*) implies (*p* implies *q*)	(chain 7 and 5; *QED*)

The proof is longer than either of the two given at the beginning of the paper. In terms of LT's methods it takes three steps instead of two or one: a forward chaining, a detachment, and a substitution. This leads to the not-surprising notion, given human experience, that length of proof is an important variable in determining total effort: short proofs will be easy and long proofs difficult, and difficulty will increase more than proportionately with length of proof. Indeed, all the one-step proofs require 500 to 1500 primitives, while the number of primitives for two-step proofs ranges from 3000 to 50,000. Further, LT has obtained only six proofs longer than two steps, and these require from 10,000 to 90,000 primitives.

The significance of length of proof can be seen by comparing Fig. 7, which gives the proof tree for (2.17), with Fig. 4, which gives the proof tree for (2.45), a two-step proof. In going one step deeper in the case of (2.17), LT had to generate and examine many more subproblems. A comparison of the various statistics of the proofs confirms this statement: the problems are roughly similar in other respects (*e.g.*, in effort per theorem considered), hence the difference in total effort can be attributed largely to the difference in number of subproblems generated.

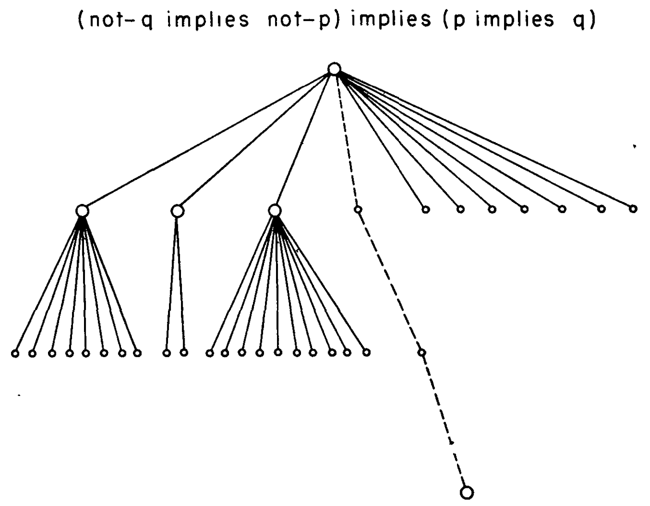


Fig. 7—Subproblem tree of proof by LT of 2.17 (all previous theorems available).

Let us examine some more evidence for this conclusion. Fig. 8 shows the subproblem tree for the proof of (2.27) from the axioms, which is the only four-step proof LT has achieved to date. The tree reveals immediately why LT was able to find the proof. Instead of branching widely at each point, multiplying rapidly the number of subproblems to be looked at, LT in this case only generates a few subproblems at each point. It thus manages to penetrate to a depth of four steps with a reasonable amount of effort (38,367 primitives). If this tree had branched as the other two did, LT would have had to process about 250 subproblems before arriving at a proof, and the total effort would have been at least 250,000 primitives. The statistics quoted earlier on the effectiveness of subproblem generation support the general hypothesis that the number of subproblems to be examined increases more or less exponentially with the depth of the proof.

The difficulty is that LT uses an algorithmic procedure to govern its generation of subproblems. Apart from a few subproblems excluded by the type II errors of the similarity test, the procedure guarantees that all subproblems that can be generated by detachment and chaining will in fact be obtained (duplications are eliminated). LT also uses an algorithm to determine the order

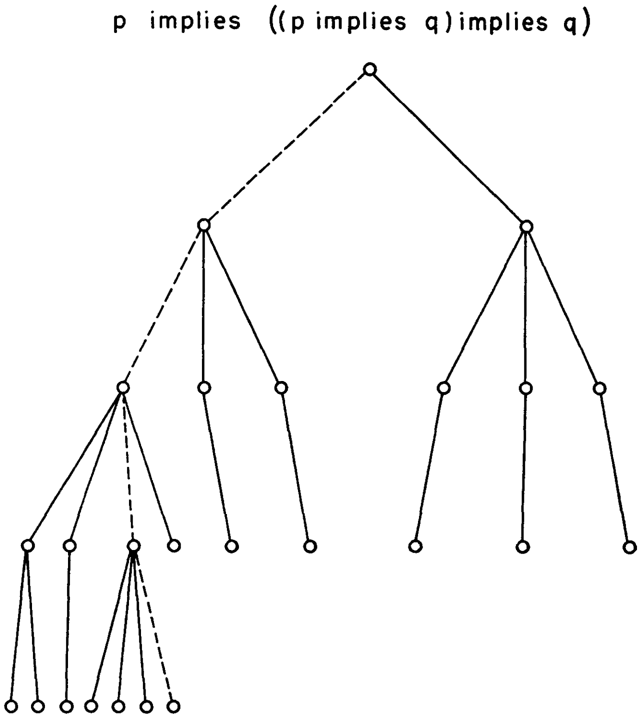


Fig. 8—Subproblem tree of proof by LT of 2.27 (using the axioms).

in which it will try to solve subproblems. The subproblems are considered in order of generation, so that a proof will not be missed through failure to consider a subproblem that has been generated.

Because of these systematic principles incorporated in the executive program, and because the methods, applied to a theorem list averaging 30 expressions in length, generate a large number of subproblems, LT must find a rare sequence that leads to a proof by searching through a very large set of such sequences. For proofs of one step, this is no problem at all; for proofs of two steps, the set to be examined is still of reasonable size in relation to the computing power available. For proofs of three steps, the size of the search already presses LT against its computing limits; and if one or two additional steps are added the amount of search required to find a proof exceeds any amount of computing power that could practically be made available.

The set of subproblems generated by the Logic Theory Machine, however large it may seem, is exceedingly selective and rich in proofs compared with the set through which the British-Museum algorithm searches. Hence, the latter algorithm could find proofs in a reasonable time for only the simplest theorems, while proofs for a much larger number are accessible with LT. The line dividing the possible from the impossible for any given problem-solving procedure is relatively sharp, hence a further increase in problem-solving power, comparable to that obtained in passing from the British-Museum algorithm to LT, will require a corresponding enrichment of the heuristic.

Modification of the Logic Theory Machine

There are many possible ways to modify LT so that it can find proofs of more than two steps in a way which has reason and insight, instead of by brute force. First, the unit cost of processing subproblems can be substantially reduced so that a given computing effort will handle many more subproblems. (This does not, perhaps, change the "brute force" character of the process, but makes it feasible in terms of effort.) Second, LT can be modified so that it will select for processing only subproblems that have a high probability of leading to a proof. One way to do this is to screen subproblems before they are put on the subproblem list, and eliminate the unlikely ones altogether. Another way is to reduce selectively the number of subproblems generated.

For example, to reduce the number of subproblems generated, we may limit the lists of theorems available for generating them. That this approach may be effective is suggested by the statistics we have already cited, which show that the number of subproblems generated by a method per theorem examined is relatively constant (about one subproblem per seven theorems).

An impression of how the number of available theorems affects the generation of subproblems may be gained by comparing the proof trees of (2.17) (Fig. 7) and (2.27) (Fig. 8). The broad tree for (2.17) was produced with a list of twenty theorems, while the deep tree for (2.27) was produced with a list of only five theorems. The smaller theorem list in the latter case generated fewer subproblems at each application of one of the methods.

Another example of the same point is provided by two proofs of theorem 2.48 obtained with different lists of available theorems. In the one case, (2.48) was proved starting with all prior theorems on the theorem list; in the other case it was proved starting only with the axioms and theorem 2.16. We had conjectured that the proof would be more difficult to obtain under the latter conditions, since a longer proof chain would have to be constructed than under the former. In this we were wrong: with the longer theorem list, LT proved theorem 2.48 in two steps, employing 51,450 primitives of effort. With the shorter list, LT proved the theorem in three steps, but with only 18,558 primitives, one-third as many as before. Examination of the first proof shows that the many "irrelevant" theorems on the list took a great deal of processing effort. The comparison provides a dramatic demonstration of the fact that a problem solver may be encumbered by too much information, just as he may be handicapped by too little.

We have only touched on the possibilities for modifying LT, and have seen some hints in LT's current behavior about their potential effectiveness. All of the avenues mentioned earlier appear to offer worthwhile modifications of the program. We hope to report on these explorations at a later time.

Conclusion

In this paper we have provided data on the performance of a complex information-processing system that is capable of finding proofs for theorems in elementary symbolic logic. We have used these data to analyze and illustrate the difference between systematic, algorithmic processes, on the one hand, and heuristic, problem-solving processes, on the other. We have shown how heuristics give the program power to solve problems in a reasonable computing time that could be solved algorithmically only in large numbers of years. Finally, we have assessed the limitations of the present program of the Logic Theory Machine and have indicated some of the directions that improvement would have to take to extend its powers to problems at new levels of difficulty.

Our explorations of the Logic Theory Machine represent a step in a program of research on complex information-processing systems that is aimed at developing a theory of such systems and applying that theory to such fields as computer programming, and human learning and problem solving.

Discussion

L. D. Yarbrough (No. American Aviation): Have you made an attempt at finding some set of theorems which might tend to optimize the proof of the remaining theorems?

Mr. Newell: No, we have not. One of the interesting things in mathematics is once an area has been studied very thoroughly metamathematical theorems are developed. One of these is called the Dix Theorem, which is a theorem about the use of all the theorems to prove new ones. It would be excellent if the machine would discover such a powerful theorem. A learning program should be developed so that the machine will learn to use those theorems which have worked in the past. We have done some experiments in this direction and we expect to report on these results.

Lt. Col. Bryan Cowan (U. S. Army): Does the LT machine select and operate on subproblems in a predetermined order, such as taking first those characterized by high probability of giving a solution?

Mr. Newell: At the moment, the routine used takes the subproblems in order of generation and this leads to the large trees. So, in fact, the machine does a large amount of searching. The logic theory machine is being used to make a much better selection but we have not gone very far in this direction. At the mement we think that this is one of the major defects.

G. H. McClurg (Signal Corps): Can the machine disprove theorems or recognize when it has disproved a theorem which it is trying to prove?

Mr. Newell: This is related to the notion of familiar theorems, which the machine does not recognize. We believe that we can devise a rule which will really throw all the false theorems out. We can use truth tables, for instance, but these would prove to be rather expensive so we are using instead a fairly cheap kind of a test which is to say that expression is most likely false. For example, if there are no common variables, then the "theorem" is probably false. I believe that I . . .

P. E. Tanner: Have you tried using the contradictory method of solution, *i.e.*, assume the negative of the proposition and prove this false?

Mr. Newell: No, we have not used this as a method for the reason that this is a poor technique.

Programming the Logic Theory Machine*

A. NEWELL† AND J. C. SHAW†

Introduction

A COMPANION paper[1] has discussed a system, called the Logic Theory Machine (LT), that discovers proofs for theorems in symbolic logic in much the same way as a human does. It manipulates symbols, it tries different methods, and it modifies some of its processes in the light of experience.

The primary tool currently available for studying such systems is to program them for a digital computer and to examine their behavior empirically under varying conditions. The companion paper is a report of such a study of LT. In this paper we shall discuss the programming problems involved and describe the solutions to these problems that we tried in programming LT.

The aims of this paper are several. First, it serves to amplify and make more precise its companion paper. Second, progress in research on complex information processing demands a heavy investment in technique It is not sufficient simply to specify a rough flow diagram for each new system and to program it in machine code on a one-shot basis. We hope this paper not only shows the techniques and concepts we found useful, but also emphasizes the role played by flexible and powerful languages in making progress in this area.

Finally, LT is representative of a large class of problems which are just beginning to be considered amenable to machine solution; problems that require what we have called heuristic programs. A description of the problems encountered in LT may give some first hints about the requirements for writing heuristic programs.

Nature of the Programming Problem

To avoid too much dependence on the companion paper, we will repeat a few general statements about LT in the context of programming. LT is a program to try to find proofs for theorems in symbolic logic. In this type of problem, a superabundance of information and alternatives is provided, but with no known clean-cut way of proceeding to a solution. These situations require "problem-solving" activity, in the sense that one has no

* This paper is part of a research project being conducted jointly by the authors and H. A. Simon of Carnegie Institute of Technology. All of us have shared in the development of most of the ideas in the language.

† The RAND Corp., Santa Monica, Calif.

[1] Newell, Shaw, and Simon, this issue, p. 218.

path to the solution at the start, except to apply vague rules of thumb, like "consider the relevant features." Playing chess, finding proofs for mathematical theorems, or discovering a pattern in some data are examples of problems of this kind. Occasionally, as in chess, one can specify simple ways to solve the problem "in principle"—given virtually unlimited computational power—but, in fact, limitations of computing speed and memory make such exhaustive procedures inadmissible.

LT, as an example of a heuristic program, may be expected to yield some clues about constructing this type of program. Actually, LT is still very simple compared to the complexity in learning, self-programming, and memory structure that seem necessary for more general problem solving. Thus, we think that LT underestimates the flexibility and programming power required in complex problem-solving situations.

Perhaps the most striking feature of LT when compared with current computer programs is its truly nonnumerical character. Not only does LT work with other symbols besides numbers, but many of its computations either generate new symbolic entities (*i.e.*, logic expressions) that are used in subsequent stages of solution, or change the structure of memory. In contrast, in most current computer programs, the set of entities that are going to be considered (the variables and constants) is determined in advance, and the task of the program is to compute the values of some of these variables in terms of the others. Such forward planning is not possible with LT. Although there are fixed entities in LT—which remain constant over the problem and provide a framework within which the computation takes place—these are complex affairs, rather than symbols. An example of such an entity is a list of subproblems. The elements on this list are variable: each problem is a logic expression which is generated by LT itself and may carry with it various amounts of descriptive information. The number, kind, and order of these logic expressions are completely variable.

The program of LT is also very large. There are large numbers of different features under consideration and large numbers of special cases. All of these features and cases require special routines to deal with them, and, by a kind of compounding rule, the existence of numerous subroutines requires yet other subroutines to integrate them. This is further compounded in LT, because no one way of proceeding ensures solution of a given logic problem, and hence, many alternative subroutines exist. Their existence again implies routines to choose among them. Some reduction in the total size of the program is achieved through multiple use of routines, but this increases the complexity of the subroutine structure considerably. The hierarchies of routines become rather large: 13 or 14 levels are common in LT.

Another characteristic of LT is its use of information about the workings of the program—how much memory is being used for particular purposes, and how much effort is allocated to various subprocesses—to govern the further course of the program. LT uses such information in its "stop rules," by which it passes from one problem to another, and in its choice between recomputing and storing information. It is cheaper in terms of total amount of computation to compute information and then store it; LT does this as long as memory space is available. When memory becomes scarce, LT shifts to recomputing information each time it is needed.

LT also contains routines for recording the results of its operation, so that we can study its behavior. It is built to permit easy and rapid change of program, in order to let us study radical program variations. These additional features do not add anything qualitatively to the features mentioned above, but they do add to the total size and complexity of the program.

Requirements for the Programming Language

We can transform these statements about the general nature of the program of LT into a set of requirements for a programming language. By a programming language we mean a set of symbols and conventions that allows a programmer to specify to the computer what processes he wants carried out.

Flexibility of Memory Assignment:

1) There should be no restriction on the number of different lists of items of information to be stored. This number should not have to be decided in advance; that is, it should be possible to create new lists at will during the course of computation.

2) There should be no restriction on the nature of the items in a list. These might range from a single symbol or number to an arbitrary list. Thus, it should be possible to make lists, lists of lists, lists of lists of lists, etc.

3) It should be possible to add, delete, insert, and rearrange items of information in a list at any time and in any way. Thus, for example, one should be able to add to the front of a list as well as to the end.

4) It should be possible for the same item to appear on any number of lists simultaneously.

Flexibility in the Specification of Processes:

1) It should be possible to give a name to any subroutine, and to use this name in building other subroutines. That is to say, there should be no limitation on the size and complexity of hierarchies of definitions.

2) There should be no restriction on the number of references in the instructions, or on what is referenced. That is, it should be possible to refer in an instruction to data, to lists of data, to processes, or what not.

3) It should be possible to define processes implicitly; *e.g.*, by recursion. More generally, the programmer should be able to specify any process in whatever way occurs naturally to him in the context of the problem. If the programmer has to "translate" the specification into a fixed and rigid format, he is doing a preliminary processing of the specifications that could be avoided.

4) It should be unnecessary to have a single integrated plan or set of conventions for the form of information; that is, for symbols, tags, orderings, in lists, etc.

On the other hand, it should be possible to introduce conventions locally within parts of the problem whenever this will increase processing efficiency.

These requirements are neither precise nor exhaustive. Except in a world where all things are costless, they should not be taken as general programming requirements for all types of problems. They characterize the kinds of flexibility we think are needed for the sorts of complex processes we have been discussing.

Solutions of the Program Language Requirements for LT

The requirements stated above for LT were met by constructing a complete language, or pseudo code, which has the power of expression implied by the requirements, but which the computer can interpret. A first version of the language was developed independently of any particular computer and was used only to specify precisely a logic theory machine.[2] A second version is an actual pseudo code prepared for use on the RAND JOHNNIAC,[3] and it is this version that we will describe here. We have had about fifty hours of machine computation using the language and hence, we can evaluate fairly well how it performs.

The present language has a number of shortcomings. It is very costly both in memory space and in time, for it seemed to us that these costs could be brought down by later improvement, after we had learned how to obtain the flexibility we required. Further, the language does not meet the flexibility requirements completely. We will comment on some of these deficiencies in the final section of this paper.

The language is purely a research tool, developed for use by a few experienced people who know it very well. Thus a number of minor rough spots remain. Further, we used available utility routines, fitting the format and symbols of the language to a symbolic loading program which already exists for JOHNNIAC. This loader accepts a series of subroutines coded in absolute, relative, or symbolic addresses (symbolic within each routine separately) and assigns memory space for them.

Description of the Language

The description of the language, which we shall call IPL, falls naturally into two parts. First, we shall describe the structure of the memory and the kinds of information that can be stored in it. Then we shall describe the language itself and how it refers to information, processes, and so on.

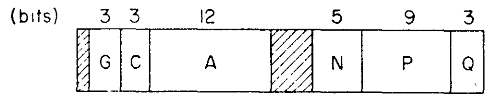
The Memory Structure

LT is a program for doing problems in symbolic logic. Basically, then, IPL must be able to refer to symbolic logic expressions and their properties. It must also be able to refer to descriptions of the expressions which are properties only in an extended sense. For example, an expression may have a name, or it may have been derived in a given fashion, or by using a certain theorem, and IPL must be able to express these facts. LT needs to consider lists of expressions and lists of processes used to solve logic problems, and there must be ways to express these facts.

Elements: The basic unit of information in IPL is an *element*. An element consists of a set of symbols, which are the values of a set of variables or attributes. There are different kinds of elements to handle the different kinds of information referred to above. The two most important elements are the logic element, which allows the specification of a symbolic logic expression, and the description, which is a general purpose element, used to describe most other things, and which carries with it its own identification.

Each element fits into a single JOHNNIAC word of 40 bits. The symbols are assigned to fixed bit positions in the word, so that the element is handled as a unit when it comes to moving information around, etc. Each variable and symbol has a name which is used in IPL to refer to it. The name of a symbol is the address of a word that contains the appropriate set of bits. Since JOHNNIAC has instructions corresponding to the logical "and" and complementation, the name of a variable is the address of a word that holds the mask necessary to extract the bit positions corresponding to the variable.

Logic elements are the units from which logic expressions are constructed. Fig. 1 shows what variables and



G Number of negation signs
C Connective (or variable)
A Location of logic expression
N Name
P Position number
Q Level in expression

Fig. 1—Logic element.

symbols comprise a logic element. Expressions in symbolic logic are much like algebraic expressions: each element consists of an operation (called a "connective" in logic) or a variable, together with the negation signs (if any) that apply to it. We use a parenthesis-free notation, in which the position of each element in a logic expression is designated by a number—this number, therefore, being one of the symbols in the element. For example, the logic expression $p \rightarrow (-q \vee p)$ would be represented by five elements as shown in Fig. 2. Each logic element consists of six variables (each taking on a variety of values) all of which fit into a single word: the number of negation signs, the connective, the loca-

[2] A. Newell and H. A. Simon, "The logic theory machine," IRE Trans., vol. IT-2, pp. 61–79; September, 1956.
[3] The JOHNNIAC is an automatic digital computer of the Princeton type. It has a word length of 40 bits with two instructions in each word. Its fast storage consists of 4096 words of magnetic cores and its secondary storage consists of 9216 words on magnetic drums. Its speed is about 15,000 operations per second.

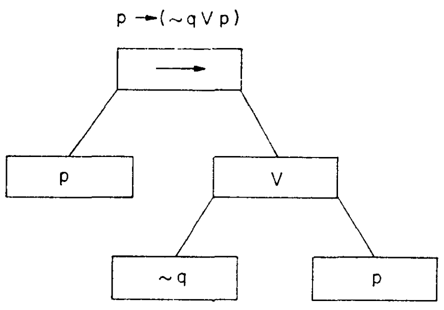


Fig. 2—Logic expression.

tion of the list which holds the entire logic expression of which this is an element, the name of the variable, the position number, and the number of levels down from the main connective.

Description elements consist of two symbols, as shown in Fig. 3. There are many different types of descriptions

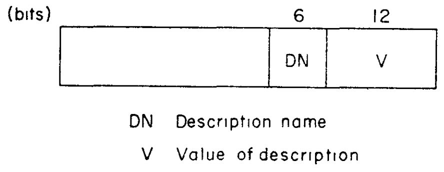


Fig. 3—Description element.

such as the name of a logic expression, the method used to derive a logic expression, or the number of different variables appearing in a given logic expression, and each type has a name. The left-hand symbol in the element gives the name of the type of description. The right-hand symbol gives the value of this description for some logic expression with which this description is associated. Thus, for example, in considering a certain logic expression the description element 012-L082 might be found. The 012 indicates that this description element gives the method used in deriving the expression, and the L082 is the name of the actual method used, in this case, the method of detachment.

Lists: Lists are the general units of information in the memory. A list consists of an ordered set of items of information. Any item on a list may be either a list or an element, and these are fundamentally different types of units, as we shall see later (the difference arises mostly from the fact that an element is contained in a single JOHNNIAC word). Since a list is itself an ordered set of items which may themselves be lists, we obtain most of the flexibility we desire in the memory structure. There is no limit to the complexity of the structures that can be built up, provided that one knows how to use them, except the total memory space available. Also, there is no restriction to the number of lists on which an item can appear. For example, if we have a list of items, we can construct one or more indexes (lists) on each of which an arbitrary subset of the items of the original list appears.

With each item located in a given list we may associate descriptive information without disturbing the general structure of the lists. That is, each item can have a list of description elements associated with it. As many descriptions may be put on the list as desired, and, since they are self-identifying (by means of the description names they contain) they may be put on in any order. Descriptions are associated with the item *on a given list;* hence, if an item is on several lists, it can have several distinct description lists.

Forming Lists: This memory structure has most of the flexibility that we specified earlier as desirable. There are information processes that can create new lists at any time; or that can add items to a list at any time, either in front, in back, or in some relation to other locatable items in the list. Likewise, items can be deleted from lists at any time, or moved from one list to another, or simply "adjoined" to a new list without being deleted from the old one.

All this flexibility in the memory is achieved by the single expedient of divorcing the ordering relations among items of information from the ordering relations built into the address structure of the computer memory. Let us sketch how this is done in JOHNNIAC for IPL.

To form a list, we use a set of *location words,* each containing two addresses. One address locates an item on the list, the other address locates the next location word. Fig. 4 shows how this is done for a list of three elements.

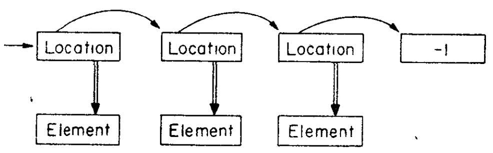


Fig. 4—List of elements. The left half of the location word contains the address of the next location word (single arrow); the right half contains the address of the element (double arrow).

A location word holding a negative number serves to terminate the list. Since the JOHNNIAC word holds two instructions and hence, contains two addresses, it is very convenient for this scheme. The left address is the address of the next location word, the right address is the address of the item on the list. In order to permit the general list structure indicated earlier, each location word contains a code telling whether the item it refers to is an element (001), in which case it contains information, or a list (000), in which case it is the beginning of another list; *i.e.*, of a series of location words. Fig. 5 shows a general list containing both elements and lists.

Each item on a list is uniquely determined by one of the location words. To associate a description list with this item, we insert a location word for the description list right behind the location word of the item with which it is to be associated. We use a code 002 to distinguish the location word of the description list from the location word of the next item. Since a description requires only half a word to hold its two attributes, we put the location of the next description in the list in the other half of the same JOHNNIAC word (Fig. 6).

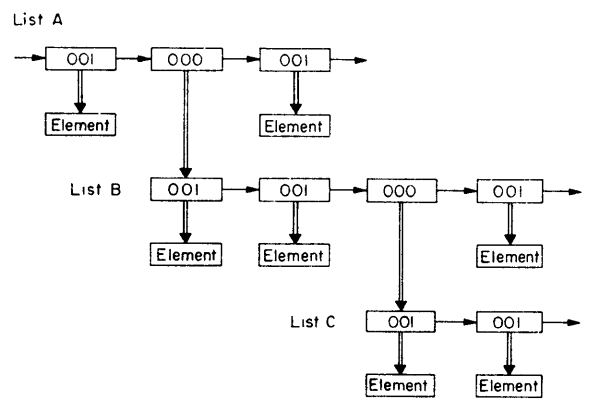


Fig. 5—General list. List B is the second item on list A; list C is the third item on list B.

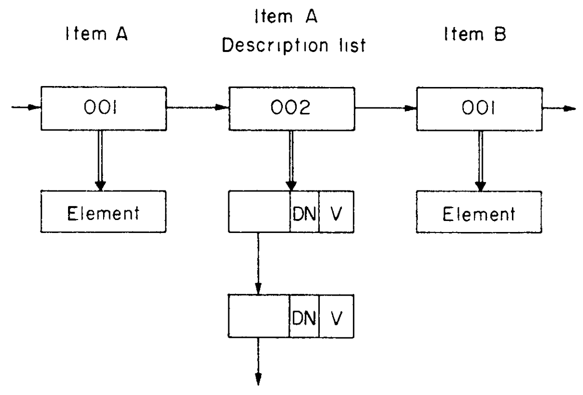


Fig. 6—Description list. The description list for item A is inserted immediately behind item A, and distinguished from the next item, B, by a 002 location word.

The address of the next item or location word in a list need bear no particular relation to the address of a given location word; they need not be adjacent, for instance. Hence, an item is deleted from a list simply by deleting its location word. Suppose, as in Fig. 7, we have three

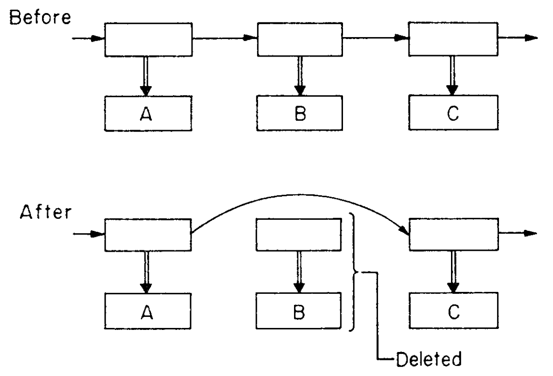


Fig. 7—Deletion of an item from a list. Item B deleted by changing address in location word of A to refer to location word of C.

items, A, B, and C, on a list. To delete B, we simply change the address in the location word of A to refer to the location word of C. Because of this same freedom of position of the words in a list, location words on different lists may hold the address of the same item of information. Hence, a single item of information may be on as many lists as we please.

Perhaps the major problem in creating a flexible memory is the housekeeping necessary to make available unused words after they have become scattered all through the memory because of repeated use and reuse. When a word is deleted from a list, we must be able to "recapture" this word, in order that it may be used subsequently for other purposes. The association memory (the name we use for this type of memory) starts with all "available space" on a single long list, called the *available-space list*. Whenever space is required for building up a new list, this is obtained by using the words from the front of the available-space list, and whenever information is erased and the words that held it become available for use elsewhere, these words are added to the front of the available space list. In Fig. 7, the deletion process would be completed by tying all of the deleted words into a list and attaching this at the front of the available-space list. Thus, the fact that unused space is scattered all through the memory creates no difficulty in finding new space, for there is a single known word (the head of the available-space list) that always contains the address of the next available word. Hence, the use of the memory is not complicated by any natural ordering like the natural sequence of machine addresses.

Since all lists obtain their new space from the same list, the only restriction on amounts or degrees of complexity of lists is the total size of memory. Thus, it is clear why there are no separate limits to the number of lists, their maximum size, how "stacked" up they can be, and so on. In this sense the language is easy to learn and use.

Language Structure

The basic form of the language is the same as in all current programming languages. The terms of the language are *instructions*. Each instruction specifies a complete information process; that is, it can be followed by any other instruction. Thus, the syntax of the language is basically identical with that of machine codes or flow diagrams: sequences of instructions are carried out in succession, with conditional transfers of control to permit alternative subsequences to be carried out as a function of the process. (IPL is slightly more general than this, as will be seen subsequently.) Also, as is usual in this general type of language, each instruction specifies separately: 1) an operation and 2) the information upon which it operates.

In IPL, a *program—e.g.*, LT—is a system of *subroutines*. Each subroutine is a sequence of instructions. Each IPL instruction is defined by a subroutine (more precisely, each particular occurrence of an instruction has its operation part carried out by some subroutine), usually called the *defining subroutine* of the instruction. Subroutines may be written either in JOHNNIAC machine language or in IPL (whenever "routine" is used in this paper it always means IPL routine, unless stated otherwise). Correspondingly, there are two kinds of IPL instructions; *primitives*, whose defining subroutine is written in machine language, and *higher instructions* whose defining subroutine is written in IPL.

The system of subroutines is organized in a roughly hierarchical fashion. There is a "master routine," each instruction of which is defined by another routine; the instructions in these subroutines, in turn, are defined by yet other subroutines, and so on. Eventually, primitive instructions are reached, and their defining subroutines, which are in machine language, are executed.

Instructions: Fig. 8 shows a typical instruction for-

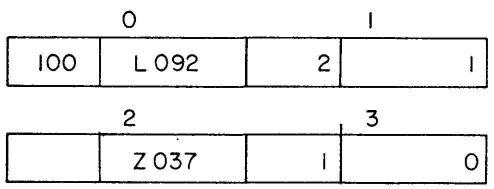


Fig. 8—Typical IPL instruction. The small numbers over the half words give the number of the reference place.

mat. An instruction is a vertical sequence of JOHNNIAC words; a routine is a vertical sequence of instructions. Each half-word in an instruction is a *reference place*, the first being numbered 0, as shown in the figure. There may be any number of reference places in an instruction, and the number need not even be constant from one use of the instruction to another, provided that the subroutine which carries out the operation understands how to use the references. Each reference states (by code): 1) the *type of reference* (the small space on the left side of each reference) and 2) the *reference*. All references are to elements; hence, to refer to a list in an instruction, it is necessary to refer to an element that refers to the list.

There are three types of references (coded 0, 1, 2). Type 0 gives the location of an element in memory by specifying either the absolute or relative address of the word containing the element. Type-0 references are used for the fixed names of things, like constants (Z037) or subroutines (L092). Within each routine, instructions are located by symbolic addresses (such as * 32) which are also of type 0. In this scheme there is no general way to make reference in one routine to an arbitrary instruction in any other routine, although there are some important special ways of referring from one routine to another which are considered below.

Each subroutine has its own working storage, consisting of an indefinite number of elements. These are referred to by type-1 references: 1-0, 1-1, 1-2, · · · . When a subroutine is completed, these working-storage elements are automatically erased and made available for reuse.

As stated above, each subroutine carries out the operation for the instruction it defines, called the higher instruction of that subroutine. Since this higher instruction has variable references that differ with each occurrence of the instruction, some way must exist of referring to these variable values within the defining subroutine. This is done by the type-2 references. The symbols 2-0, 2-1, 2-2, · · · , in a subroutine refer to the reference places 0, 1, 2, · · · , of the higher instruction defined by the subroutine. Thus, the type-2 references are indirect, referring to an element by referring to a reference place, that, in turn, refers to the element. The situation is shown in Fig. 9, where a given routine, L081, uses an

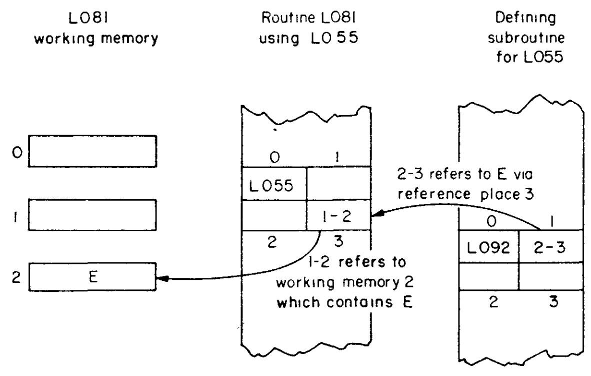

L081 working memory
Routine L081 using L0 55
Defining subroutine for L055
0
1
2
E
0
1
L055
1-2
2
3
2-3 refers to E via reference place 3
1-2 refers to working memory 2 which contains E
0
1
L092
2-3
2
3

Fig. 9—IPL Type 2 reference.

instruction, L055, which is a higher instruction. Thus, L055 is defined by a subroutine, part of which is shown further to the right. In the working memory of L081, shown at the far left, the element E is located in cell 2. The instruction L055 refers to E by using symbol 1-2 in reference place 3. The instruction L092, which is in the defining subroutine of L055, refers to E by 2-3.

The first reference place (number 0) in an instruction determines the operation; or, more precisely, refers to the subroutine that will carry out the operation. All other reference places may refer to anything needed by the subroutine. Thus, an instruction is simply a format for a general process that is a function of an arbitrary number of variables.

Execution of Instructions: Access is gained to the subroutine that defines an operation by reference to an element which contains the location of that subroutine. These elements are normally collected in a directory (the Lxxx region), but may be put on lists and processed like other elements.

From the point of view of coding for JOHNNIAC, the language is entirely interpretive. When the interpreter picks up an IPL instruction, it obtains the address of the directory element from reference place 0. Besides giving the location of the defining subroutine, the directory element tells whether the instruction is a primitive or a higher instruction.[4]

In case an instruction is a primitive, the defining subroutine is in machine language, and the interpreter transfers control to it. This subroutine then either moves the referenced elements into fixed positions or adapts its instructions to the addresses of the referenced elements, and carries out the operation. Upon finishing it returns control to the interpreter.

The directory element also gives other information which is described in the section, "Other Details."

In the case of a higher-level instruction, the defining subroutine is also written in IPL and requires further interpretation. To interpret the subroutine, the interpreter sets up several lists (obtaining space for these from the available space list). The first list contains the referenced elements in the instruction; it is the "2" list from the point of view of the subroutine. The second list is the "1" list, which will hold the working memory elements, as they are set up in the subroutine. Finally, before beginning to interpret the subroutine, the interpreter must add to the next-instruction list the location of the instruction following the one it is currently interpreting.

Within the subroutine the interpreter picks up the first instruction and repeats the process described above. Thus, no matter how many levels there are in the hierarchy, the interpreter continues to set up the lists described above for each successive subroutine until it reaches a primitive instruction. After the primitive is executed, the interpreter proceeds to the next instruction in the lowest subroutine. When this subroutine is completed, the interpreter backs up to the next lowest subroutine, and so on. In operation, the memory structure for interpretation looks like a gigantic yoyo: lists of references are set up successively one "below" another as the interpreter goes down in search of a primitive, and then these lists are erased again in reverse order as the routines they correspond to are finished.

Primitive Processes

So far we have described only the outline of a language—the structure of memory and the format of the instructions. The power of the language to express complex processes depends on the set of primitive processes out of which all the others must be built.

The set of primitives in IPL is built to reflect the principle that the programmer should need to know as little as possible about the storage of information in memory. One of the clear lessons from programming experience is that small differences in what the programmer must know about the information in memory have important consequences for ease of programming. Much of the power of automatic computation derives from the fact that in order to program it is sufficient to know only the location of a number, and not the number itself. Further, large gains in programming efficiency have come from allowing the programmer to know this location only as a symbol or a relative address, rather than as an absolute address.

In IPL an attempt was made to carry this principle one step further. The concept of *working memory*, already encountered earlier, is used to divide the memory into two parts, so that all the intricate processing is done in working memory. The remaining memory, which we shall call the *list memory*, is used for permanent storage of information. This division of memory separates the primitive operations into two groups. One group of operations finds information in the list memory, makes it available in the working memory, and stores it back in the list memory again. The other group of operations processes information in working memory. There are also primitive operations for input and output, which will be discussed in the next section.

Working-Memory Operations: The primitives for processing information in working memory are roughly similar to typical machine instructions for a two-address computer. An example will make this clear. Fig. 10

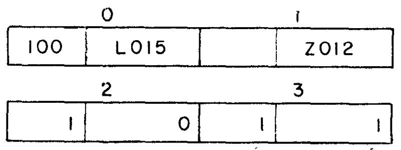


Fig. 10—IPL addition instruction.

shows a typical occurrence of L015, the addition instruction. The instruction adds a value stored in working memory 1–0 to a value in working memory 1–1. Since a working memory holds an entire element, which is a collection of attribute values, it is necessary to indicate which attribute is being added; the Z012 in reference place 1 designates this. Z012 is the name of an attribute: in this case the number of negation signs of a logic element. Hence, this instruction reads, "add the number of negations in the element 1–0 to the number of negations in element 1–1, and place the result in 1–1." This type of instruction requires the programmer to know what information is in the working memory elements and defines some elementary process involving two of them.

The set of primitives for processing information in working memory includes addition and subtraction instructions; test instructions for equality and inequality with a conditional transfer of control to some other part of the subroutine; and instructions for copying information from one working memory to another. All of these instructions use a reference, like the Z012 in the example, to designate which attributes in the element are being considered.

Find and Store Operations: The find and store instructions, which pass information between list memory and working memory, are quite different in nature from the instructions discussed above. To avoid having the programmer know anything in detail about the location of information in the list memory, all the find and store instructions take the form of searches through a list with tests to identify the information desired.

An example will make this clear. Referring back to Fig. 8, L092 is a primitive find instruction that obtains information about a logic expression. A logic expression is stored as a list of elements (see Fig. 2) in the list memory. The order of symbols in a logic expression is specified by position numbers and is unrelated to the ordering of the elements in the list. Given the position number of a logic element it is easy to compute the position number of the element that is in any given relative position to it, say, its left subelement. L092, then, is an instruction that finds an element in a logic expression which bears a specified relative position, (*e.g.*, Z037) to some element (*e.g.*, in 2-1) already known, and that puts it in a working memory (*e.g.*, 1-0) where it can be processed further. Thus, the programmer only has to know that the element he wants bears a given relation to some known element, and he need know nothing about the actual location of this element in the list or about the rest of the logic expression. Each logic element carries as one attribute the location of the list of the logic expression containing it, so this does not have to be found separately. Typically, when an element is called for by an instruction, it is not known whether the desired element even exists; hence, L092 provides a conditional transfer of control if the desired element is not found. This particular instruction is written as a primitive because the programming problem it solves—to find a logic element bearing a given relation to a known logic element—occurs repeatedly in LT.

The instructions for finding descriptions provide a second example of how the instructions concerned with the list memory use search and test processes. As stated earlier, a list of description elements can be associated with any item in a list. An instruction to find a description requires the programmer to know the item to which the description applies. The programmer must also know the name of the description he wants. The operation then searches the list for the item, and when it finds it, searches the description list associated with that item for the description with the indicated name. Again there is no guarantee that the item is on the list, that the description is on the description list, or even that a description list exists; and the failure to find the desired description is signaled with a conditional transfer of control.

Like the find instructions, none of the store instructions depend on the precise location of an item in a list. A typical store instruction is L023, which moves descriptions from working memory to the description list of a known item on a known list. L023 searches the list until it identifies the item, then searches down the description list until it identifies the description name of the description it is storing. If it finds it, it stores the new value; if it does not find it, it stores the description as a new item on the description list. L023 must also be prepared to set up a description list in case it does not find one at all. One of the important features of the descriptions is that no space needs to be reserved for them until they are actually created.

Other Processing Instructions: Besides find and store instructions for the various types of lists, there are instructions for erasing lists, for creating lists, and for moving items from one list to another directly. There is no erasing problem in the working memory, since working memory elements are erased automatically when a subroutine has been carried out. In erasing items from lists, the instructions require only that the programmer know what item is to be erased and on what list it occurs, but not its location on the list. Likewise, the programmer does not have to know anything at all about the structure of a list to erase it, but only where it starts. The erase operations are constructed to explore all possible extensions of a list and erase them all.

Other Details

No attempt has been made with this language to build a repertoire of service routines or to make input and output exceptionally convenient. For output, the JOHNNIAC has either punched cards or a high-speed numeric printer, but we use the printer almost exclusively. There is a "print list" primitive, which prints any list however complicated and extensive. This single primitive essentially suffices for our output needs, since, if we have several lists we wish to print, we simply put them on a new superordinate list in the right order, and apply the "print list" instruction to this superordinate list. The instruction then prints out the several lists in the indicated order. We can suppress all the location words, so that only the items of information print.

JOHNNIAC has punched-card input. We use a card format for giving an arbitrary list to the computer, so that a single "read list" primitive suffices for data input. The program input is handled by the symbolic loading routine mentioned earlier.

The use of the interpretive mode for the language allows the computer easy access to its own process. As a matter of course we trace the IPL instructions that are being performed. The trace can be selective, each directory element indicating whether the trace of that instruction is to be printed or not. What is printed is the name of the subroutine (*i.e.*, the relative address of the directory element) indented according to its level in the hierarchy of routines. Since we wish to study the course of the processing as well as end results, the trace is a prime source of data.

Also as a matter of course, we keep tallies of the number of times each instruction is performed, both for our use as data and for the program's use in operating. The directory element also tells the address of the tally. For example, LT allocates its effort by using such tallies to see how much effort it has devoted to a given problem.

The devices mentioned above provide us with some debugging facilities. Since all the information connected with the hierarchy of routines is on lists (see the section on the language structure), we can print a single debugging list which contains these plus a number of other lists as items. The printing of this list (with all location words being printed) gives us most of the information we need. We also use the tracing with a selective suppression of details to aid in debugging. This procedure traces all instructions within the subroutines of interest, and none of the instructions in those of no interest.

The JOHNNIAC's 4096 words of high-speed, random-access core storage is not adequate for a program and data lists of this size. LT in operation has about 1600 words of interpretive code, about 1600 words of machine code, and about 400 words of directories, constants, etc.; hence, a total storage of about 3600 words for the program alone. We have been forced to utilize secondary storage, which for JOHNNIAC, is a drum of 9216 words. Storage hierarchies are notorious for presenting difficult problems of accessibility, and the type of program we are working with, with its avoidance of consecutive blocks of words, simply compounds the the difficulties. So far, we have used the drum only for the program, and not for data; we are keeping almost all the higher routines on it.

When the interpreter goes to the directory element of a given instruction, it discovers whether the defining subroutine is in cores, or on the drum. If the subroutine is on the drum, it is fetched into the next available stretch in a large consecutive block in core storage. As the interpreter works down the hierarchy, more and more subroutines are brought in from the drum and gradually fill up this large block. Each subroutine remains intact until it is finished, but no attempt is made to plan or schedule trips to the drum. As soon as a subroutine is completed it is "discarded" and the next routine from the drum is placed in the same stretch of the core storage block.

Evaluation of the Language

The previous section has given a picture of the solutions we tried in programming LT. We will now consider more critically what this language accomplishes, and what its shortcomings are.

Association Memory

We have made a great issue of the flexibility of memory—the ability to create lists at will and to add and delete items from existing lists. This has certainly simplified a number of housekeeping tasks. For instance, the entire structure involved in the hierarchy of subroutines with their indefinite numbers of working memories was easily handled by means of the association memory. Similarly, in a primitive like "erase list," which must search out all items in a list of arbitrary structure, there is a need to remember an indefinite number of junctions in exploring the list. The flexible memory allows the primitive to build up a list of these points of choice, adding each new one to the front of the list.

We have made extensive use of the flexibility throughout LT, the one major program we have written in IPL. Our most complicated structure to date is a list of lists of lists connected with a routine that modifies the list of theorems used by LT as a function of experience. This same structure also has theorems (a list of logic elements) as items on multiple lists.

The association memory also has severe costs. The most obvious cost is the extra memory space needed for location words. Location words occupy about one half of the list memory, since it takes one location word to refer to each "item" word in a simple list. The proportion of location words is not much greater than one half, since the space devoted to simple lists greatly exceeds the space devoted to the more complicated structures that take additional location words. This cost factor is rather difficult to estimate, however, since alternative schemes for achieving the same total program are not known. Any component comparison is somewhat misleading, since the virtues of the association memory arise from the avoidance of planning, of reserving blocks of storage, and so on.

Another cost, which may be the more serious one, is the loss of ability to compute addresses. In a computation which can be well laid out in advance, it is often possible to assign addresses to data in such a way that the addresses can be computed in a simple fashion. For example, instead of searching a table for a function value corresponding to a given argument, the address of the function value can be made a simple function of the argument, say the argument plus a constant, and the value obtained almost without effort. This is not possible with the association memory, where the only function the address can perform is to designate the location of another word in a list.

The Language Structure

Some of the flexibilities of the language structure have provided greatly increased power in the language whereas others have not. We have not made much use of the variable number of reference places if one measures use in terms of variability of that number. Most of our instructions have about four references: the operation and three pieces of information. Both examples described in this paper are of this size. Whenever a routine exceeds about six references—one of the executive routines has 15—the references are not used as "variables" but to transmit data. In the case of the executive routine, for example, the 15 references provide a convenient place to hold all the parameter values for a run of LT. On the other hand, we have used the variable number of references considerably as a flexible communication device up and down the hierarchy of routines. Thus, in making changes in the program it is often convenient to transform what was a constant into a variable. This can be done simply by adding a new reference place to the higher instruction and replacing the constant by a type-2 reference, say 2-6, if the original instruction previously had only references 0 through 5.

We have used extensively the hierarchical properties of the language—the ability to define new subroutines in terms of old ones. The number of levels in the main part of LT is about 10, ignoring some of the recursions, which sometimes add another four or five levels. It would be interesting to compare the size of the LT program written in IPL and the program written in machine code. This is very difficult to do, since when writing in machine language one makes use of subroutines, and even of subroutines of subroutines. Hence there is no standard machine language program for comparison. However, the following figures give a rough approximation. IPL consists of about 45 primitive instructions, which take an average of about 70 JOHNNIAC instructions each. Instructions are packed two to the JOHNNIAC word, so the number of words used is roughly 35 per primitive. In addition the machine-language subroutines all include some initial code either to position the words used by the subroutine, or to adapt its instructions to the addresses of the words. This can be an appreciable fraction of some of the simpler primitives like L015, the addition instruction. Further, these statistics do not reflect the fact that the primitives themselves use a number of closed subroutines.

The LT program described in this and the companion paper contains about 45 different higher instructions, defined by 45 higher routines. A typical higher routine contains about 16 primitives and two higher instructions. If we expand the entire hierarchy for LT, ignoring recursions, we find that LT can be written as about 8000 primitives. Since the average primitive instruction takes about two JOHNNIAC words to write, it is clear that some hierarchization of subroutines is needed to compress a program like LT into manageable size.

The fact that the operation part of an instruction is a reference place like all the others, and can be treated as such, gives additional power to IPL. An operation is normally referred to by its "name," which is the relative address of the directory element that leads to the defining subroutine; *e.g.*, L015, L092, etc. However, an operation can also be referred to by a type-1 reference, such as 1-3, if the correct element is in the working storage. For instance, LT uses a set of routines, called methods, which are, roughly speaking, alternatives to one another, and are used in about the same way. There is a list of methods, which is simply a list whose items are the directory elements of the methods. The executive routine executes a method by searching the list until it finds the desired one, bringing it into a working memory (*e.g.*, 1-3) and then performing an instruction with 1-3 in the 0 reference place. If this method does not work, the executive routine finds the next method and repeats the process. Thus the executive routine is able to perform a simple iteration over the set of methods. We use this device also to compute sets of descriptions of logic expressions.

We can also use a type-2 reference for an operation. This essentially makes the operation a variable and dependent on information in the higher routine. This device is used in several places in LT, but only to allow fixed specification at a higher level. We have no examples where the operation is determined by a computation in the higher routine, although this is possible.

An entirely different kind of power arises from the flexibility of the hierarchy—the ability to do recursions. An instruction may be used in its own defining subroutine, or in any of the subroutines connected with its definition, in any way whatsoever provided that the routine does not modify itself and that the entire process terminates. The restriction on self-modification is clearly needed if the same routine is to be available at more than one level. All the information necessary to carry out the routine must be stored in the working memory, which is set up separately for each occurrence of the routine, and not within the routine. In LT there are no higher routines that modify themselves. The impetus for self-modification of routines usually arises from the use of iterative loops. In LT all iterations are accomplished by means of lists. A succession of elements is brought in from a list to fixed working-memory references, and the iteration terminates when the end of the list is reached.

There are two kinds of recursions in LT. The matching routine, which compares one logic expression with another, is an example of the first kind. The routine starts with the main connective of the expression and proceeds recursively down the tree of the expression element by element (see Fig. 2). The recursion is bound to stop, since the number of elements in any expression is finite. This recursion could also be expressed as an iteration through the list of the expression, although perhaps not so neatly.

A more fundamental recursion occurs at the highest levels of the program. Here LT has an executive routine which governs its whole problem-solving behavior. Within this routine, that is, at some lower level, are methods that generate subproblems. Also within this routine are subroutines that select the subproblem to be worked on next. A subproblem does not differ from the original problem with respect to the methods and techniques used to solve it. Hence the appropriate programming technique is to apply the entire executive routine to the subproblem; that is, to perform a recursion with the entire program. Such a recursive system will terminate if a solution is found, but since no guarantee exists that the problem will be solved there is no guarantee the machine will stop. In LT we add such a guarantee simply by having LT stop after a certain total amount of effort, a rather trivial but effective device.

The language also has its drawbacks. It is expensive; the over-all average time for a primitive is about 30 milliseconds. JOHNNIAC performs an add order in about 80 microseconds. Thus if we consider L015, the addition instruction, and compare it with a direct replication of its operation in machine language, we find we lose a factor of about 60. This is one of the more extreme cases. If we consider an instruction like L092, which is typical of the list operations, the loss factor drops to about 5. However, as in the case of the association memory, a component comparison is somewhat misleading, since all the virtues of the interpretive scheme arise from its automatic handling of the entire problem. For example, the hierarchy provides a way of keeping track of some 50 words of data in process, and it would seem that this information must be maintained if the problem is handled in any other way. The appropriate comparison is with an alternative way of coding a total problem such as LT, and no comparable alternative currently exists.

The large hierarchy with its multiple levels may seem a very expensive feature. However, its cost appears to be less than the cost of interpreting the primitives, primarily because of the infrequency of higher routines in comparison with the number of primitives. All the higher instructions account for only about 10 per cent of the total number of instructions interpreted, whereas the unit cost of interpretation of a higher instruction is only two and a half times as great as for a primitive (about 50 ms to 20 ms). Thus interpretation of all the higher routines accounts for less than 30 per cent of the total cost of interpretation.

Additional Deficiencies of IPL

Experience in writing programs in IPL has revealed a number of additional deficiencies. Perhaps the one that strikes the programmer most is the artificiality of the distinction between the element and the list. By packing a set of symbols into a single JOHNNIAC word we gain in memory space over schemes that use one full word for each variable. The net result, however, is that certain properties, those packed into an element, are treated in one way, and others, those expressed by the lists or by the description elements, are treated in another. Elements are brought into working storage for processing; since lists have various sizes and shapes, they cannot be handled in this fashion. Information that must be kept as a list is handled by indirect reference, through an element in working storage that refers to it. Information that can be fitted into an element is handled directly in working storage. For example, an element and a one-element list must be processed very differently in IPL.

A second deficiency is the restriction to certain forms of referencing. IPL has great flexibility in the specification of operations, that is, an operation can be specified by giving an expression in the language for that operation. We have allowed no such flexibility in the specification of the other references. There are only three ways of giving the information to be used in a routine: by giving the address of the element, the name of the working storage that holds the element, and the name of a reference place that refers to the element. These methods allow certain indirect references, but they still lack flexibility. A rather simple example, but one that is typically annoying, occurs when we want to refer to a name of a routine, that is, to a symbol like L082, which is the address of a directory element. This symbol is used in many places throughout the program, but there is no simple way of getting to it. There is no reason why there should be less power of expression for information references than for operations. It should be possible to give a reference by giving an expression for determining that reference, just as is now done in IPL for operations.

There are other unsolved problems. For instance, we have no satisfactory way of erasing in the association memory. The problem is not how to delete items and make their space available again, which we think is done fairly well in IPL. The problem is how to know what can be erased, since there is no direct way of knowing what else in the system may be referring to the items about to be erased. References are directional, so that if location word A refers to item B, there is no way of knowing this, when only the address of B is known. Uniform two-way referencing seems to be an expensive solution, although it may be the only one. In simpler programs this erasing problem is handled by having the programmer know at all times exactly what refers to what. But if we move to programs in which all lists are set up during operation by the program itself, such solutions are not adequate, and the problem soon becomes acute.

Conclusion

IPL is an experimental language that was built to find ways of achieving extreme flexibility. It was developed in connection with a particular substantive problem—proving theorems in symbolic logic—which requires great flexibility in the memory structure, and powerful ways of expressing information processes.

The language achieved its purpose: we have a running program for LT which has allowed us to explore its behavior empirically with a number of variations. On the other hand, the language is relatively crude, viewed as a general language for specifying programs like LT. It is very costly; it shows the "provincialism" of too close a connection with symbolic logic; and it still has a number of rigidities.

We believe that the basic elements of the language are sound, and can be used as the ingredients of languages having considerably greater powers of expression and speed. We are currently engaged in the construction of a new language patterned on IPL, which we hope will serve us as a general tool for the construction and investigation of complex information processes.

Discussion

L. P. Meissner (Nol, Corona): Do you have a list of those lists which do not list themselves?

Mr. Shaw: Without going further into paradoxes except to say that there is not a direct answer to this question, but the debugging list does list itself.

P. Sayre (Northrop): Would you reiterate or expand your remarks on the next version especially with regard to automatic programming?

Mr. Shaw: No, except to suggest that programming itself is a field of complex information. Processing such is the field we are studying.

J. Matlock (Douglas): Can you give an example of a subroutine using itself?

Mr. Shaw: I think the best example of this is the matching routine which is asked to match one expression to another. The first part of this routine merely looks at the main connectives. If it is successful in matching the given expression to the second one, it then takes a look at the lower left element of each expression and there again it is faced with exactly the same problem as it was faced with initially. Again the matching routine is asked to match this expression. So, at this point the routine recourses and calls upon itself to match the expression it is faced with to the second expression. Eventually, of course, it comes to the termination on these trees and proceeds to back off. So, it says, "I am done" to itself, reiteratively, and then backs up to a certain point at which it proceeds down the right branch.