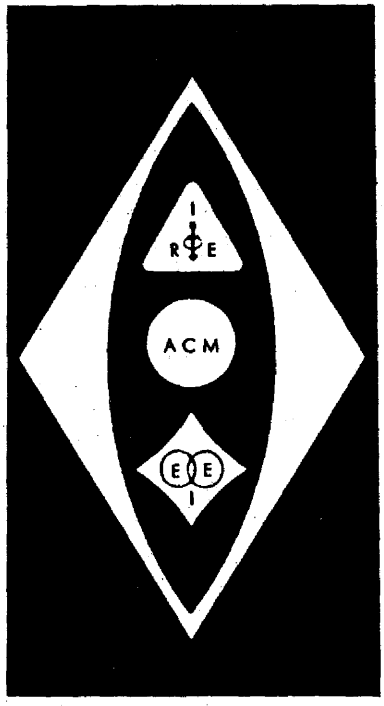
Proceedings of the

EASTERN JOINT COMPUTER CONFERENCE

December 13-15, 1960 New York, New York



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

Vol. 18 Price $3.00

PRIOR NJCC CONFERENCES

Number	Conference	Location	Date
1	Eastern	Philadelphia	Dec. 10-12, 1951
2	Eastern	New York City	Dec. 10-12, 1952
3	Western	Los Angeles	Dec. 4-6, 1953
4	Eastern	Washington	Dec. 8-10, 1953
5	Western	Los Angeles	Feb. 11-12, 1954
6	Eastern	Philadelphia	Dec. 8-10, 1954
7	Western	Los Angeles	Mar. 1-3, 1955
8	Eastern	Boston	Nov. 7-9, 1955
9	Western	San Francisco	Feb. 7-9, 1956
10	Eastern	New York City	Dec. 10-12, 1956
11	Western	Los Angeles	Feb. 26-28, 1957
12	Eastern	Washington	Dec. 9-13, 1957
13	Western	Los Angeles	May 6-8, 1958
14	Eastern	Philadelphia	Dec. 3-5, 1958
15	Western	San Francisco	Mar. 3-5, 1959
16	Eastern	Boston	Dec. 1-3, 1959
17	Western	San Francisco	May 3-5, 1960

PROCEEDINGS OF THE EASTERN JOINT COMPUTER CONFERENCE

PAPERS PRESENTED AT
THE JOINT IRE-AIEE-ACM COMPUTER CONFERENCE
NEW YORK, N. Y., DECEMBER 13-15, 1960

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

EASTERN JOINT COMPUTER CONFERENCE

ADDITIONAL COPIES

Additional copies may be purchased from the following sponsoring societies at $3.00 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.

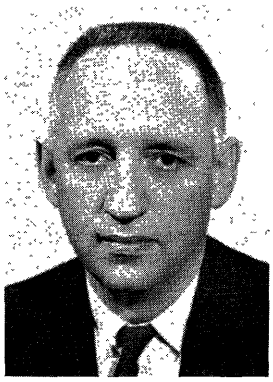
AMERICAN INSTITUTE OF ELECTRICAL ENGINEERS
33 West 39th Street, New York 18, N. Y.

ASSOCIATION FOR COMPUTING MACHINERY
14 East 69th Street, New York 21, N. Y.

Manufactured in the U.S.A. by the Fifth Avenue Lithographic Associates, Inc., New York, N. Y.

Harry H. Goode

July 1, 1909–October 30, 1960



Harry H. Goode, Professor of Electrical Engineering at the University of Michigan and a prominent leader in the activities of the National Joint Computer Committee and several societies active in the computer field, died in an automobile accident on the morning of October 30, 1960. His loss will be deeply felt by all who knew him through his teaching, his frequent lecture appearances, his many publications, his work in professional societies, his consulting activities, his stimulating participation in conferences, or directly through his warm friendship.

Professor Goode was born in New York City on July 1, 1909. He received the B.S. degree in history from New York University in 1931, and later earned the Bachelor of Chemical Engineering degree from Cooper Union in 1940 and the M.A. in Mathematics from Columbia University in 1945. His early professional work was in statistics, and in 1941 he became Statistician-in-Charge for the New York City Department of Health. During the war years he was a research associate at Tufts College and worked on applications of probability to war problems and also on the acoustic torpedo problem. From 1946 through 1949 he was on the staff of the Office of Naval Research at the Special Devices Center, Sands Point, Long Island. Here he progressed through successive responsibilities to be head of the Special Projects Branch. His work during this period was on flight control simulation and training, aircraft instrumentation, anti-submarine warfare, weapon system design, and computer research. Through his O.N.R. work he was actively associated with such pioneering computer projects as the Whirlwind computer at M.I.T., the Cyclone computer built by Reeves Instrument Company in New York, and the Typhoon computer built by R.C.A. Laboratories for the Navy.

In 1950 he joined the Willow Run Research Center of the University of Michigan, serving first as head of the Systems Analysis and Simulation Group, next as Chief Project Engineer, and then as Director of the Center. Under his direction the Research Center carried forward a broad program of research, including system design, computers, radar, infra-red, and acoustics, and in the process doubled its size to 600 people. He guided the efforts of the Center through problems in air defense and battle area surveillance, and was instrumental in establishing the basis for the ground system for the Bomarc missile.

In 1954 he was appointed Professor of Electrical Engineering at the University of Michigan, and in 1956 his wide range of interests brought a dual appointment as Professor of Electrical Engineering and as Professor of Industrial Engineering. In 1958 he served for a year as Technical Director of the Systems Division of the Bendix Corporation, maintaining a fractional appointment in the University so that he could continue to teach his newly introduced course on System Design. A little over a year ago he returned to full-time teaching and research activities in the Department of Electrical Engineering.

In addition to his wide range of services to the University of Michigan, Professor Goode served as a consultant to industry and government, and was active in professional society affairs. He brought to problems a keen insight and a rare ability for stripping away the non-essentials. His advice was highly valued and widely sought. Among the firms for which he consulted were the United Aircraft Corporation, the Bendix Corporation, the Auerbach Electronics Corporation, the DuPont Corporation, the Ford Motor Company, the Burroughs Corporation, the Texas Company, and the Franklin Institute. He served the government on projects of the National Bureau of Standards, the Post Office Department, the Air Force, and the House of Representatives Appropriations Committee. For the Air Force, he was chairman of the W-117L Committee on Advanced Reconnaisance; and for the House Committee, he served as a member of the Study Group on Missile Reliability.

He served his profession as a member of the Administrative Committee of the I.R.E. Professional Group on Electronic Computers from 1953 to 1956, as a member of the Computer Advisory Committee of the Society of Automotive Engineers, and as a member of a subcommittee of the A.I.E.E. Committee on Feedback Controls. His most important service in this area was as chairman of the National Joint Computer Committee of I.R.E., A.I.E.E., and A.C.M. In this latter role, he played an important part in the formation and formulation of the charter of the International Federation of Information Processing Societies.

Professor Goode was a member of many societies — The Association for Computing Machinery, the American Mathematical Society, the Mathematical Association of America, and the Institute for Mathematical Statistics. He was a Fellow of the American Association for the Advancement of Science and a senior member of the Institute of Radio Engineers. He was also a member of Sigma Xi, Eta Kappa Nu, and Mu Alpha Omicron.

His many published papers touched upon statistics, simulation and modeling, vehicular traffic control, and system design. His major published work is the book "System Engineering," of which he was senior author with R. E. Machol. The book was an outgrowth of the very successful and valuable course which he introduced at the University of Michigan under the title, "Large Scale System Design."

Professor Goode's broad experience with computers and his participation in national computer functions led to his participation as one of the group of eight Americans who visited Soviet computer establishments in 1959.

Our profession has lost one of its most outstanding members—a man of rare versatility, talent, vigor, and vision.

FOREWORD

This volume contains the papers presented at the 1960 Eastern Joint Computer Conference (the eighteenth Joint Computer Conference). In order to make the Proceedings available at the conference these pages were reproduced directly from the authors' manuscripts by photo offset.

The papers which are presented here were selected from among 130 that were submitted. On the basis of 1,000-word summaries Elmer Kubie and his Program Committee selected those which seemed of exceptional significance, originality, timeliness and interest.

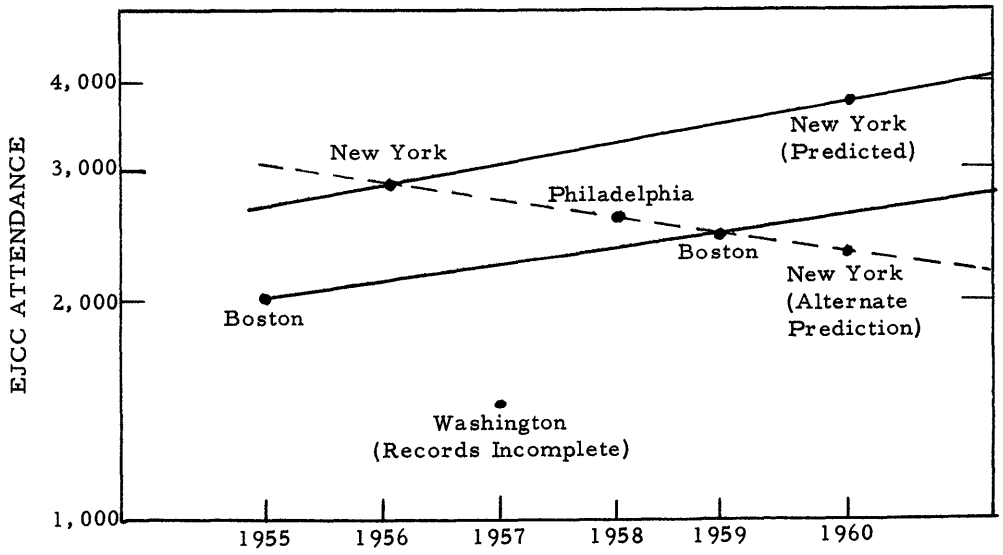
A study of the records indicates that when judged by the box office, the programs of the EJCC seem to fill a need. The following graph shows the attendance at recent meetings and predicts the 1960 attendance from the known growth in Boston, assuming that the growth in New York would be at the same rate. The graph also shows an alternate interpretation of the data according to which there is no geographical effect and really the situation is deteriorating. If the first interpretation is correct, there is no hotel in New York that can hold the EJCC. This is the reason for the choice of the combination of the Hotel New Yorker and the Manhattan Center Auditorium. However, there is hope for a better future since the projected Americana West Hotel will be large enough.

When the Joint Computer Conferences are judged by the critics' notices rather than by the box office a different picture emerges. Many of these critics think that the primary benefit of such a conference is the opportunity to meet ones friends (or competitors) in the halls and lobbies to exchange views and to pass on the latest inside information. At this conference we heeded their advice and made an effort to assist this kind of communication.

After each session there was a discussion period of a new kind. There was some space available at the rear of the auditorium, and in this space each speaker was stationed at a particular spot so that people could ask him questions. These spots were to serve also as focal points for the gathering of groups of people whose interests were aroused by the papers. Not only could they talk to each other and to the speaker but also they had the opportunity to form luncheon and dinner groups of people with congenial interests.

For nearly a year the committee members have worked with me on preparations for the conference. I take this opportunity to thank them for the many hours of hard work and the large contribution they have made.

Nathaniel Rochester
General Chairman



NATIONAL JOINT COMPUTER COMMITTEE

Chairman

Mr. H. H. Goode *
Department of Electrical Engineering
University of Michigan
Ann Arbor, Michigan

Vice Chairman

Dr. Morris Rubinoff
Moore School of Engineering
University of Pennsylvania
Philadelphia, Pennsylvania

Secretary-Treasurer

Miss Margaret R. Fox
National Bureau of Standards
Department of Commerce
Connecticut Ave. & Van Ness St.
Washington 25, D. C.

IRE Representatives

Mr. Harry H. Goode '60 - '61
Department of Electrical Engineering
University of Michigan
Ann Arbor, Michigan

Mr. Frank E. Heart '60 - '61
Lincoln Laboratories
Rm. B-283
Post Office Box 73
Lexington 73, Mass.

Dr. Willis H. Ware '59 - '60
The RAND Corporation
1700 Main Street
Santa Monica, California

Dr. Werner Buchholz '59 - '60
IBM Product Dev. Laboratory
P.O. Box 390
Poughkeepsie, New York

AIEE Representatives

Dr. Morris Rubinoff '60 - '61
Moore School of Engineering
University of Pennsylvania
Philadelphia, Pennsylvania

Mr. R. R. Johnson '60 -'61
Computation Laboratory
General Electric Co.
Phoenix, Arizona

Mr. Claude A. R. Kagan '59 - '60
Engineering Research Center
Western Electric Company, Inc.
Box 900
Princeton, New Jersey

Mr. Stanley Rogers '59 - '60
c/o Convair, Box 1950
Mail Zone 7-08
San Diego 12, California

ACM Representatives

Mr. Paul Armer '60 - '61
The RAND Corporation
1700 Main Street
Santa Monica, California

Mr. Walter W. Carlson '60 - '61
Design Division
E.I. du Pont de Nemours
Wilmington 98, Delaware

Mr. J. D. Madden '59 - '60
System Development Corp.
2500 Colorado Avenue
Santa Monica, California

Dr. H. R. J. Grosch '59 - '60
415 East 52nd Street
New York, New York

Ex-Officio Representatives

Dr. Harry D. Huskey
University of California
Department of Mathematics
Berkeley, California

Mr. R. A. Imm
International Business Machines Corporation
Rochester, Minnesota

Dr. Arnold A. Cohen
Remington Rand UNIVAC
St. Paul 16, Minnesota

Headquarters Representatives

Dr. J. Moshman
C.E.I.R.
1200 Jefferson Davis Highway
Arlington 2, Virginia

Mr. R. S. Gardner
Assistant Secretary
American Institute of Electrical Engineers
33 West 39th Street
New York 18, New York

Mr. L. G. Cumming
Technical Secretary
The Institute of Radio Engineers
1 East 79th Street
New York 21, New York

* Deceased

EASTERN JOINT COMPUTER CONFERENCE COMMITTEE

General Chairman	Nathaniel Rochester, IBM Corp.
Assistant to General Chairman	Robert J. Haughey, IBM Corp.
Program	Elmer Kubie, Chairman, Computer Usage Co. Dr. Julius Aronosky, Socony Mobil Oil Co. George R. Briggs, RCA Laboratories Charles Doersam, Potter Instruments Felix Kalin, International Telephone and Telegraph Co. Roy Reach, Minneapolis-Honeywell Regulator Co. Fred Warden, International Telephone and Telegraph Co. Daniel M. McCracken
Publications	Clem J. Rachel, Chairman, Remington Rand Univac, Division of Sperry Rand Corp. Noel K. Zakin, Remington Rand Univac, Division of Sperry Rand Corp. Richard T. Kanter; Remington Rand Univac, Division of Sperry Rand Corp.
Publicity	Jack Heaney, Chairman, Sylvania Electric Products, Inc. James Lanigan, Sylvania Electric Products, Inc.
Exhibits	Alan D. Meacham, Chairman, Gille Associates Donald A. Boell, Gille Associates
Registration	Jack Behr, Chairman, Packard Bell Computer Corp.
Hospitality	Robert P. Fopeano, Chairman, Bendix Computer Division of the Bendix Corp.
Hotel	Robert J. Williams, Chairman, IBM Corp. W. W. Ward, IBM Corp.
Finance	A. I. Schott, Chairman, The National Cash Register Co.
Local Arrangements	Benjamin W. Leavitt, Chairman, General Telephone & Electronics Laboratories Allen L. Brown, New Canaan Research Center, Inc.

TABLE OF CONTENTS

(Continued)

TABLE OF CONTENTS, continued

A LOGICAL MACHINE FOR MEASURING PROBLEM SOLVING ABILITY

Charles R. Langmuir
The Psychological Corporation

Summary

The magnitude of costs incurred by assigning unsuccessful or even marginal personnel to tasks involving EDP systems design and programming justifies a much greater effort in the selection of personnel than the use of conventional aptitude tests implies. A small desk-top machine named the Logical Analysis Device is described, its logical organization is explained, and its operation as a method of observing and testing an individual's problem solving abilities is illustrated. Some comment describing the wide variation of performance among several hundred college graduates employed in various professions is included but the principal emphasis is given to data pertaining to the performance characteristics of persons in computer and data processing activities. The application of the device is clearly indicated at the point of evaluating final candidates for assignment to tasks requiring a high order of logical and analytical talent.

* * * * *

The talents, interests and aptitudes of individuals who become effective computer programmers are probably basically similar in all the many varieties of EDP installations. In making this statement, I do not mean to suggest that all persons who are happy, successful, contributing workers in the computer profession are all alike. Any such notion is patently absurd. I do mean to indicate that there are certain essential characteristics which are common among persons who are able to live peacefully, in comfort, and perhaps in joy, with modern computing machinery and the extraordinary variety of problems in which the machinery becomes involved. A principal purpose of this paper will be the amplification of the idea in the opening sentence including a statement of what the fundamental characteristics of successful computer programming personnel are, and a description of a method of observing, indeed, even measuring, an individual's status with respect to these characteristic abilities.

When a computer installation is established in a university environment, individuals who like this kind of thing seem to gather around it. They simply gravitate to their center of attraction. After a time, and often quite a long time, they either weed themselves out or they get into a suitable orbit. To a less obvious degree, the same kind of self-selection of computer personnel takes place in a scientific computing center, including perhaps computer installations in industrial organizations which are primarily concerned with computing and data processing in the so-called scientific categories.

In the business-type organization where the computer installation is primarily concerned with the processing of commercial paper work and reports, the development of the personnel situation is somewhat different. There may be an initial surge of enthusiastic interest when the decision to install a computer is first announced, but this is only superficially comparable with the gravitation of personnel characteristic of scientific institutions.

The difference between the university or scientific-type installation and the commercial or business-type installation becomes apparent in examining the effects of the weeding-out process. In the business data processing operation, weeding out of ineffective personnel is accompanied by much difficulty and all the pain that abnormal personnel readjustments call forth in business organizations. In addition to the organizational disruptions that occur, there are very large dollar costs involved. These costs quickly become great enough to justify the attention I will suggest should be given to the initial selection of personnel.

When the personnel department calls you, the supervisor, to announce that another candidate has appeared for the opening in the programming department, an important and costly decision is implied. When you hire your man for training in this activity, you commit the organization to an investment not less than $5,000, more likely $20,000 and perhaps a good deal more. If, at the end of six, nine or twelve months, you find your candidate is not going to make it, the investment is a loss. If the man is marginal, a yet larger investment is required before you will find out what the return may be.

Obviously then, if it is possible to identify the characteristics that are required for successful accomplishment of the tasks involved in utilizing a computer, it is economically important to employing organizations to know what procedures can be effectively used. I shall focus my attention upon the characteristics I believe to be essential and shall present these generally with an emphasis upon data processing rather than scientific computing. The abilities I shall discuss are not those that involve spe-

cialized educational background and knowledge of particular subjects such as college mathematics or physical sciences. The academic background I postulate as necessary is only that which we accept as the common heritage of the educated person in the modern world. On the basis of experimentally observed facts, we may have to reconsider the question whether certain intellectual elements of importance in computer work are as much a part of the common heritage as we would like to believe.

What are these characteristics? First, there is certainly some minimum ability to read. Or to be more abstract, the ability to cope with verbal notation. A second requisite is ability to deal effectively with quantitative concepts and numerical notation. It is certainly no detriment to a person to be able to handle literal notation, but facility with algebraic manipulation is not included as essential. There is, third, the ability to see relations, to see order in sequences, and perhaps the ability to enrich the understanding of details by seeing analogies and abstract classifications involving order, symmetry and the permeation of common characteristics in a background of seemingly independent elements.

During the last thirty or forty years, psychologists have developed efficient ways of testing individuals for their ability in these dimensions, particularly the first two, the ability to read and the ability to handle numerical problems. It is, therefore, no problem to evaluate applicants for computer programming opportunities with respect to these abilities. We can certainly find out whether they can read well enough to handle the language in a machine manual, and we can find out if they are able to handle numbers in simple arithmetic problems. The third element, namely, the ability to see abstract relations, is not so well understood, but there are tests available. The tests I speak of are conventional paper and pencil instruments quite widely available on the professional market and well known in schools, colleges and employment offices. One such test has been specifically prepared by a computer manufacturer for use in testing applicants for training as programmers.

Such tests have proved adequate for the initial elimination of candidates. They can be economically used for screening among many applicants to eliminate those who are inadequate in verbal or numerical reasoning abilities, and probably to identify individuals whose verbal ability reaches a high level but who have difficulty dealing with the abstract kind of content or representations of a non-verbal character. There is, however, abundant evidence that such screening tests are not sufficient. Many individuals score above whatever cutting point we may choose but still lack some crucial abilities required for successful work in programming. What are these crucial elements?

Certainly one is an acceptance of the idea that systematic, logical, analytical processes can converge on a solution to problems involving complex logical relations especially in those problems containing elements of dependent serial order. A second crucial characteristic goes beyond the simple acceptance of analytical processes as a mode of problem solving but involves some minimum power in utilizing analytical procedures. Sufficient power is necessary to cope with a multiplicity of elements and an ability not only to analyze a problem into its elementary components but to synthesize the bits of information; an ability to put the bits and pieces together into a whole and to do so not by accident and not by chance but with full understanding of the ultimately closed system.

There are, of course, other desirable abilities and traits of personality -- some that we notice after the fact and that we have no success whatever in forecasting. There are the individuals who simply get ideas. Things occur to them. We do not see the mental machinery in operation and we cannot find out much about it afterwards. We call it intuitive creativity, and we are very grateful for it when it occurs. But this rare characteristic is out of reach, and I do not include it now within the domain of practical human engineering and certainly not in the incessant and mundane activity of routine personnel selection.

It is possible, however, to obtain a quite objective, very reliable estimate of a person's ability to use logical methods in solving logical problems. The procedure involved presents an individual with a logical problem, fully defined with respect to the rules of its logic; one which is simple enough to comprehend in a brief time interval, yet is complex enough to represent a real challenge, and presented in a form which makes observation of the performance not only objective, but detailed in its step by step development. By the simple device of presenting an individual with a sequence of several problems graded in a series of increasing complexity, we are able to observe both his characteristic preferences as shown by his choice of problem solving procedures and his power in synthesizing final solutions to problems.

The LAD Method

The Logical Analysis Device is a simple logical machine which can be used to observe objectively the performance of a person in manipulating logical concepts and solving logical problems. The portion of the equipment of interest is the operators display panel shown in Figure 1. The operator is the person whose problem solving prowess is being tested. The examiner, who must be a person qualified by experience and training, presents an opening explanation with demonstration. The full explanation requires ten to fifteen minutes and incorporates a carefully organized demonstration with a practice exercise as part of the familiarization program.

The dynamic elements of a demonstration with working equipment in real time cannot be simulated in any written material, but the following description does define the logical nature of the problems and suggests, at least by implication, some of the dynamic elements that are revealed in individual performance records.

In the upper left corner of the display panel, there is an indicator light labeled TIME. It is a clock which shows the passage of time in alternating intervals like day and night. The light is on for three seconds and off for three seconds and then on again and so on continuously. There are nine numbered lights arranged in a circle and one light in the center.

Next to each light in the circle, there is a push button switch. These switches are manual inputs. Each switch has the effect of turning on its associated light subject to an important restriction. Each light is either a day worker or a night worker; it can be turned on at any time during one or the other, but not both time phases. When it is turned on, it will stay on until the end of its active time phase. At the end of its active phase, it will extinguish and remain extinguished until it receives another input signal.

The target light in the center has no associated manual input switch. It can be turned on only as the consequence of some configuration of the signal lights in the circle being on. Certain crucial information about the possibilities is supplied by an information diagram. The arrows on the diagram link pairs of lights. The existence of an arrow, as the one from light 3 to X, the target light, is information that light 3 has some effect on X. The relation is not reversible, i.e., in the example illustrated the arrow from 1 to 8 states that 1 has an effect on 8 but 8 has no effect upon 1.

Any effect will occur at the end of the active time interval of the activated light and will continue to hold through the following time interval. For example, if the logical relation is simple cause-effect, then turning on number 1 in its active period will cause number 8 to come on in the following time interval. It is logically necessary that the lights at opposite ends of any arrow be active in opposite time intervals.

There are three different logical relations that may exist. The first is mentioned above--simple cause-effect - namely, turn this light on and after a while the other light will come on, and this one will go off. Obviously, any arrow can be tested by experiment to find out if it represents this effector relation.

A second relation is the combinor. If two arrows converge on one light as 9 and 2 converge on 1, neither one may be sufficient to turn on 1 but in combination they may. The existence of this combining relation can be tested experimentally, but the experimentation requires a little more planning and a little more logical sophistication to be analytically complete than does the simple try it and see experiment to prove the simplest effector relation.

The third and last relation is the preventor. A light which has a preventor relation to another negates the effects of any effector or combinor relations upon the same light. The fact that an arrow represents the preventor relation can be experimentally demonstrated but planning and correctly executing the experiment requires a greater logical precision than the tests for the other relations. In the example illustrated in Figure 1, lights 3 and 8 combine to turn on the target but 3 and 8 and 7 do not turn on the target. By such a sequence of trials, we ascertain that 7 is a preventor and the complete configuration necessary for turning on the target is 3 and 8 and not 7.

All the arrow relations can be investigated and their specific nature, i.e., which one of three, can be determined by experimental observation. In many cases the facts about a relation can be determined by logical deduction. Hypotheses may be formulated on the basis of partial information and tested.

Additional rules of the system are clarified, e.g., the existence of every relation in a problem is represented by an arrow in its diagram; an arrow represents one and only one relation; when a light goes out the machine reverts to its prior state and remembers nothing; and, all problems are soluble. Thus, the logical system is closed and completely defined. The only facts the operator does not have explicitly defined in advance are the specific relations represented by the arrows.

After the system has been fully defined and demonstrated, the problem solving task is specified in three steps. First, find out what combination of lights turns on the target, i.e., investigate the arrows to the center. Second, investigate the other arrows and thus determine what relation each represents. Third, using the information derived by logical deduction and experiment, synthesize a way of turning on the center light by some operations limited to the three red buttons, numbered 4, 5, and 6 at the bottom segment of the circle. Success on this last step is the solution of the problem. The operator, however, has complete freedom to choose his method of procedure. He may skip over the first steps as outlined if he wishes.

The operator works on the task in isolation but he has immediate access to the examiner for consultation; paper and pencil are supplied for notetaking; and he has a written summary of the rules of the system for reference.

When the solution has been attained or after a suitable time if the problem is not solved, the examiner interrupts the work and by

questioning ascertains the individual's comprehension of the logical structure of the problem. In this quizzing process, the problem is reviewed in detail, and the effectiveness of the back solution as a general method is demonstrated again.

If the problem was solved with explicit clarity of understanding of the logical relations, the examiner presents a new problem of greater complexity. If the problem was not solved or was solved without evidence that the logical structure was understood, a new problem at the same level of complexity is presented.

This elaborate procedure is carried through consistently and in as standardized a manner as possible. The function of the examiner is, in fact, that of a non-directive instructor or demonstrator. The purpose of the careful and repetitive instruction is to minimize and, if possible, eliminate any bias in the evaluation of the ultimate performance that might be caused by accidental "sets" or rigidity in persisting with an inappropriate initial choice of method. For example, an individual who is interested in probability concepts may decide that an effective approach could ignore any analysis of the information diagram as suggested by the examiner. The solution involves only three switches, and he may conclude that the possibilities are exhaustively covered by a small number of experimental trials. In such an instance, it is the task of the examiner to provide the operator with an easy opportunity to adopt a new approach. If an individual persists in using ineffective methods, we are at least able to say that his rigidity is not a consequence of lack of exposure to more effective procedures.

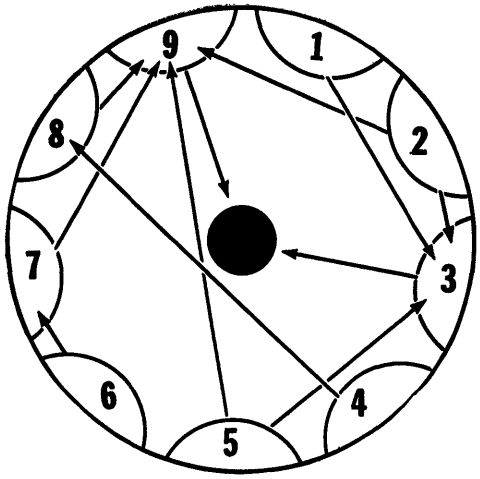
The whole process is demonstrated with the problem represented in Figure 2. By experimental trial, the operator can discover that 9 and not 3 gives X. Light 3 is a preventor. In any order that he chooses the operator can ascertain that

1 gives 3
2 gives 3 and 9
3 prevents X
4 gives 8
5 gives 3 and 9
6 gives 7
7 gives null result
8 gives null result
7 and 8 combine to give 9
9 gives X

Note the expression "can ascertain." The operator has been shown effective methods, but this fact does not mean that he will choose to use them.

With this information which represents the total logical structure of the problem, it is easy to determine that the combination 4 and 6 will initiate a sequence of events that will turn on the target light. Any attempt to use light number 5 to activate 9 directly will set up the preventor. Note also that the arrows from lights 1 and 2 represent irrelevant elements in

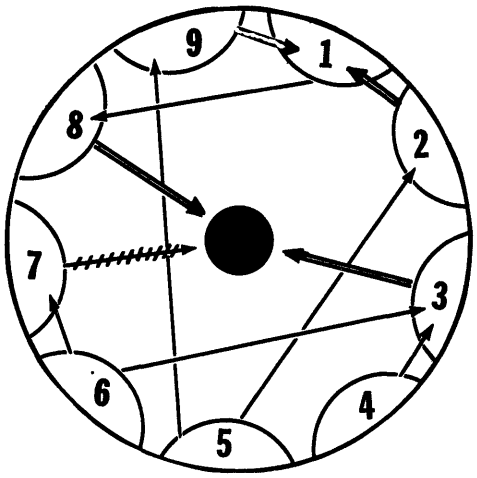
Figure 2



the logical structure. They are analogous to noise in a circuit. They are indeed logical relations, they obey all the rules, but they are irrelevant because they have no inputs other than the manual switches. Thus, if an operator makes a careful study of the information given and applies the rules of the system, he will be able to deduce that these three arrows can be ignored.

Now look at Figure 3.

Figure 3



The arrow diagram in this illustration is the same as the one illustrated on the Display Panel in Figure 1. The arrows in this diagram are coded so that a single solid line represents an effector, paired lines represent a combinor and crossed lines represent a preventor. (This information is not supplied to the operator in actual practice. It represents the in-

formation he would be able to get by experiment or deduction, or both.) The condition for the target is clearly 8 and 3 and not 7. By examining the other information, it is readily seen that light number 6 cannot be a part of the final solution. It provides a way of turning on 3 but it also turns on the preventor 7. Light number 4 turns on 3 directly. We now have half the solution, we need only to find out how to turn on light number 8. By tracing the arrows back, 8 to 1, and from 1 to the combinors 2 and 9, and from these lights back to light 5, we see that the solution will involve the operation: turn on 5, wait, and turn on 4 at the time 1 comes on.

In actual experience with this problem, the most usual first attempt at a solution involves pressing buttons 4 and 5 simultaneously. The result is not successful. The operator has to become aware of the problem of phasing his operations on the lights. In the more complex problems in the series, the operator has to get similar, but more sophisticated insights. The rules of the system are invariant, but the complexity of specific problems varies widely.

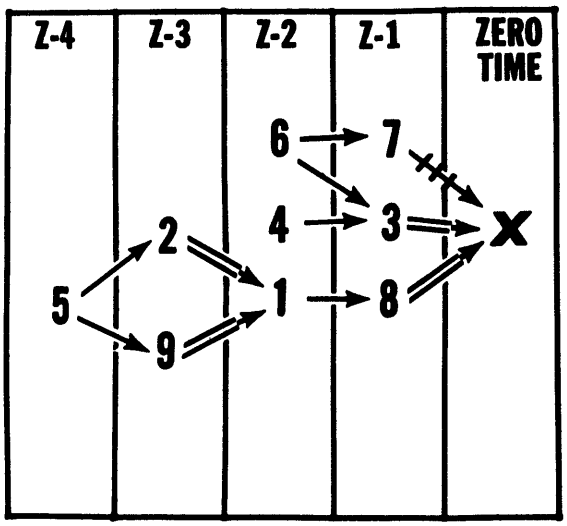
There are five levels of complexity in the complete series of problems: the two demonstrated above which are used as learning exercises and three levels beyond these.

It is an interesting demonstrable fact that with such a simple logical structure it is possible to develop complexities sufficient to differentiate among college educated adults on an ordered scale of 15 categories. The most difficult problem contains sufficient complexity to provide ample opportunity to observe the methods of work and the effective power of persons as skilled in logical performance as top-notch programmers, logical designers, and systems analysts.

Almost every operator takes notes of some kind. It is conceivable that some effects associated with the kind of notation system adopted might introduce chance variation in the performance. The procedure minimizes evaluation errors from this source by presenting a very powerful notation system to the operator after he has had the experience of working the first two problems. The standardized system is shown as in Figure 4. When initially written out, the arrows are undifferentiated. As information is verified by experiment or deduction, the arrows are coded. By logical analysis applied to this convenient reorganization of the information diagram, it is possible to derive optimum sequences of experiments that will converge on a solution.

Much repeated experience with the presentation of the notation system reveals a significant finding. A large proportion of operators do not make effective use of the recommended or any other notation. Examiners get a strong impression that taking notes is some form of academic "doodling," a kind of behavior that is approved

Figure 4



in the circumstances, whether effectively functional or not.

Evaluation of Performance

The LAD procedure incorporates a number of elements of interest in psychometric technique. The presentation is uniform, almost rigorously standardized without being formally "canned." The system strives to minimize the variability of performance attributable to the examiner's presentation. The individual's step by step performance is recorded by a remote printer. As a consequence of this technique, the operator works in isolation without any anxiety-inducing interactions resulting from the presence of an observer. The problems are real, logical structures and do not contain the tricky elements characteristic of puzzles. The increasing complexity of the series of problems is achieved without any change in the initially established logical rules of the system. Parallel forms of the problems at each level of complexity are available. It is an interesting and important fact that individual operators do not recognize parallel form problems as logically identical, even when they work them in succession. The problems are specific configurations of a completely defined logical system. Successful solution of the problems is not dependent in any way on substantive knowledge not within the experience of every educated adult.

The scoring of a completed problem-solving session on LAD leads to a rating assigned on a 15 point scale from A+, A, A-, etc., down to E+, E, and E-. This scale covers a range of performance from extremely powerful and efficient solutions to performances so ineffective that we are unable to conceive of a performance that could be demonstrably worse. The E rating indicates that

the operator was not able to achieve any success with the least complex problem after 90 minutes of repeated instruction and experience with parallel forms. The E- rating is reserved for operators who never catch on to the idea that one light may be related to another. Such a record occurs less than once in a thousand trials. The entire E category, including E and E+ ratings, represents very poor performance. About 4% of our sample of employed adults fall in this category.

The first phase of the scoring procedure is largely clerical. A count is made of the total number of operations performed on each problem. The total time worked on each problem is computed from the calibrated printed record. The individual elements of the performance are serially numbered in the order in which they appeared in time. This standardized information abstracted from the original serial record is tabulated in an organized form that enables the examiner to see at a glance the basic elements of the operator's record on each problem.

These clerical procedures reduce considerably the amount of information a rater must consider, but the amount retained has proved to be too formidable for any mathematical or mechanical computation of a final score. The rater still must consider the abstracted record and decide on the basis of all the factors present which point on the rating scale best describes the total performance. In addition to the highest level of complexity successfully handled, and the speed and the economy of effort in terms of numbers of operations, the rater will consider the approach to a major area. He will consider whether the operator's approach is logically sound. He will seek evidence that the operator grasped the import of the results of his operations. Were all the major problem areas explored? Was the order in which they were explored logical? Were many repetitions required before the operator planned his next experiment? The possibility of answering such questions about a person's problem solving efforts is a unique aspect of the LAD performance record.

The ability of the rater is central to the success of the system. The rater must be trained, must be fairly logical himself, and must have had enough experience with LAD procedure to apply the generalizations about problem solving which are contained in the ratings. It is an important, experimentally observed, fact that the subjective elements of the evaluation procedure are easily maintained in statistical control. Different examiners working independently in evaluating a single series of performance records will, of course, report different scores at least occasionally. The magnitude and variability of their differences describes the reliability of the scoring process. In many hundreds of records, accumulated over a period of three years, the differences between raters exhibit a mean of zero and a small variance. The correlation between pairs of raters evaluating the same records will be .95 or better. The same statistical results describe the comparison of ratings arrived at independently over an interval of a year or more. These findings are important in considering the validity of the LAD procedure as a method of describing individuals. The examiners are able to reach a scale of some kind of absolute judgment which does not include individual bias and does not drift with temporal effects over long or short intervals. This happy result is not the normal expectation in tasks that involve elements of subjective judgment.

Experimental Results

Table 1 shows the results of scoring the performance records of 1109 adults employed in a variety of occupations and 175 college students. For simplicity, the subdivisions of the literal categories have been grouped.

Table 1
Distribution of LAD Ratings
N=1284

x	f	%
A	211	16
B	285	22
C	518	40
D	220	17
E	50	4

The typical or median value in the sample is 8, equivalent to the letter category, C. The variation within and between the sub groups that comprise the total is large. The highest scoring group, a programming staff in an industrial scientific computer department, obtained a median score of 2, equivalent to an A rating. The lowest scoring groups obtain median scores of 11, equivalent to a D rating.

These data provide background information and nothing more. They describe the variation we can expect to observe when we test people with problems of this kind. They do not contain any evidence that performance in the miniscule problem solving situation is related to any characteristics of people working in real life situations. The possibility that important correlates may exist between behavior observed in the test situation and behavior in the real world is strongly suggested by the apparent similarities between reactions to difficulties in LAD problems and behavior in the more complex problems met in such real tasks as control engineering, designing logical circuitry, laboratory trouble shooting and computer programming.

In the LAD problems we frequently observe individuals who get "stuck in a rut." They exhibit a lack of flexibility that makes it very difficult for them to abandon an ineffective approach. Other very typical difficulties include overlooking side effects, misinterpreting data, abandoning systematic procedure under stress of frustration, ignoring the outcome of experiments

which yield null results, jumping to conclusions and assuming hypotheses are true, disregarding alternate possibilities, forgetting or distorting objectives, adopting a superficially logical but actually absurd approach, unnecessary or pointless repetition and preoccupation with redundant or even random busy work. We have all observed some of these characteristic barriers to optimum performance in ourselves occasionally and quite frequently in others. These and other elements in problem solving behavior frequently observed in LAD testing are obvious analogies to actual vocational tasks. Their existence provides a rational basis for the hypothesis that behavior exhibited by an operator's work with LAD exercises is an expression of stable, individual, personal characteristics and that these characteristics which can be observed systematically in the LAD procedure will also be characteristic elements in the individual's working environment. If the hypothesis is true, it should be possible to find differences in LAD performance for groups of people employed in real work which requires dramatically different abilities even though it is impossible to obtain reliable observation of the important component elements in the individual's performance in the job.

Experimental data which meet the requirements of dramatic difference in required abilities are presented in Table 2.

Table 2
Comparison of Programmers and Insurance Salesmen

X	Salesmen	Programmers
A	0	51
B	4	56
C	19	51
D	22	29
E	12	3
N	57	190

Both groups are of comparable age. The median programmer scores well up in the upper half of the LAD scale (Mdn=B). The typical man in the sales group scores in the lower half (Mdn=D+). Some individuals in each group have certainly made a mistake in their commitment to their vocational choice. If we were able to identify them with assurance, the difference between the groups would be larger.

The difference between the groups is not a simple difference in problem solving power. A detailed review of the records shows a striking qualitative distinction in the procedures used by most members of the sales group. There is a popular conception that workers in tasks that depend heavily upon inter-personal relations utilize some special kind of logic--a people-oriented as contrasted with a problem-oriented thinking process--sometimes thought of as intuitive and divergent as contrasted with objective, logical and convergent. On LAD this different mode of planning or decision-making procedure is observed with great frequency among people-oriented people as exemplified by sales representatives, including engineering sales, counseling psychologists and administrators in personnel management.

The typical performance of individuals in this group is broadly described as non-analytic. Almost everyone begins work on a LAD problem by seeking, more or less systematically, the configuration of circle lights that activate the target. We interpret the sequence of operations involved in this phase of the task as an analytical information-seeking mode of attack. After this basic elementary step has been accomplished, the order of operations may or may not be clearly seen to be an orderly systematic extraction of information which progressively reduces the number of unknown elements in the problem structure. When this kind of logical sequence is observed, we say the operator persisted in the analytic mode. However, such an obvious pattern may not appear. In this case, we cannot classify the mode of attack simply from knowledge of the sequential order of the experimental operations performed. The decision whether the mode is analytical or non-analytical depends upon the subsequent utilization of whatever information is retrieved, and the operator's understanding of the logical structure of the exercise at the end of the working time. Non-analytical methods of working LAD problems are accompanied by lack of precision in identifying the structural elements in the simpler problems and failure to achieve solution at the more complex levels.

If the analogues between elements of LAD performance and characteristic attributes seen in programming skill are closely similar to aptitudes that are critical for computer work, there should be an observable relation between ratings of LAD performance and supervisor's ratings of the merit of individuals who have programming responsibility. The direct experimental verification of the fact of such a relation is not as simple a matter as it would seem at first glance. Computer installations are young institutions. They differ widely in function, type of data processed, administrative organization, equipment, and experience with personnel. There are no standards for evaluation of merit on the job that are comparable from group to group, and the typical group is too small to provide within itself evidence that is reliable in the statistical sampling sense. Nevertheless, it has proved possible to obtain some correlations between LAD ratings and supervisor's rankings of individuals.

In five groups numbering 15 to 25 individuals in each, the correlation between the LAD examiner's ranking and the supervisor's ranking varied from .45 to .81. In two of these groups the individuals were ranked a second time after an interval of two years on the job. In one of

these organizations, the correlation between the supervisor's original ranking and his ranks assigned two years later was Rho=.50. The original and the follow-up correlations with LAD ranks were .70 and .74. In the other group, the original rankings supplied by a manufacturer's instructor at the end of an extended training program correlated .81 with LAD. The on-the-job ranking two years later correlated .80.

In four groups of smaller size varying from 6 to 8 members, similar correlations appear. The typical values vary around Rho=.7. Higher values may be expected in groups that range widely from excellent to inferior. Lower values are expected in homogeneous groups where the submarginal workmen have been eliminated. It has also been found that correlations are higher in groups where the ranking has been supplied by supervisors who are themselves experienced working programmers.

The fact that the rank order evaluations of performance on the job are not comparable across groups makes any attempt to use the correlation statistics in a practical regression equation rather hazardous. On the other hand, correlation findings strongly support the view that the LAD procedure could be used effectively in the practical business of selecting the most promising among applicants and also for ascertaining what proportion of an applicant group is likely to meet some minimum standards of job performance after training.

The first step in accomplishing this objective has been taken by establishing quite arbitrary, subjectively arrived at, specifications of minimum acceptable LAD performance. After analysis of the LAD record with special attention to the analytical elements characteristic of the performance, we classify the individual into one of four categories: 1. Highly recommended; 2. Recommended; 3. Marginal; and 4. Not recommended. Stated in more elaborate language, the category Highly Recommended means "This man will learn the computer rapidly. He will not have difficulty or be confused by the rigorous logical elements in understanding and utilizing machine language and machine commands. After formal instruction he will continue to learn on the job from his senior colleagues and from the day to day experience in office routine. He will advance rapidly to assume independent responsibility for substantial programming tasks. Most of the individuals who ultimately become 'creative programmers' will develop from this category." The Recommended category means about the same but with less rapid development, less efficient de-bugging, less assurance of attaining status of independent responsibility and less likelihood of becoming an outstanding contributor to the organization.

Statements in this non-quantitative job-oriented language are, of course, ambiguous to some extent, but they seem to be meaningful to supervisors responsible for computer operation. However, it is extremely important to keep in mind that such statements are forecasts of things to come if appropriate training and opportunities are made available. Since they are predictions, they may be in error. Before the classification of applicants can be acted upon automatically, it is necessary to determine the truth value of the statements.

It would be ideal to identify 1,000 persons in each category, provide the programming opportunities, and then two or three years later count heads and evaluate the work of the supervisors. Real life circumstances do not make this experiment possible and the best approximation we have been able to achieve so far are less than definitive.

SUMMARY OF FOLLOW-UP RESULTS

N	LAD	Supervisor's Report
		Installation A
6	HR	1 disappointment, 5 achieved independent responsibility
10	R	All good but surely not so good as HR's
9	NR	Do not get the idea
		Installation B
5	HR	1) slow but capable; 2) very effective; 3) high powered; 4) top man in charge; 5) can't get to know him
2	R	1) effective programmer, chief debugger; 2) effective, flexible
1	M	Effective within limits and under supervision
6	NR	All comments negative or evasive
		Installation C
8	R	All satisfactory. We decided to take no chances.
		Installation D
6	HR	All very effective
3	R	All good but not as good as the HR's
1	M	Not much imagination and slow to learn new developments
		Installation E
6	R	Estimates correct
3	M	1) marginal; 2) outstandingly effective
1	NR	Estimate correct
		Installation F
5	HR	3 outstanding, creative; 2 very good
11	R	2 outstanding, excellent; 7 good but not the best; 1 adequate; 1 mediocre
3	M	2 solidly good; 1 mediocre
5	NR	2 mediocre; 3 poor programmer

The informality and non-comparability of these supervisor's evaluations precludes the possibility of combining the results in a single table of probabilities. The data do, nevertheless, suggest the magnitude of errors of two kinds. Errors of the first kind occur when a person who is predicted as Highly Recommended proves to be a disappointment. This kind of error for the Highly Recommended and Recommended groups has occurred with small relative frequency. Errors of the second kind, namely, discovering excellent performance on the part of marginal and not recommended candidates are less well defined in the data possibly because there is greater ambiguity and greater difference of opinion and much reluctance connected with declaring disparaging evaluations.

Errors of the second kind, which reject a candidate erroneously, are of much less economic consequence to the employing organization, whereas errors of the first kind involve losses of important magnitude.

The writer concludes from the evidence so far accumulated that the effort involved in using the LAD procedure results in a significant economic pay-off.

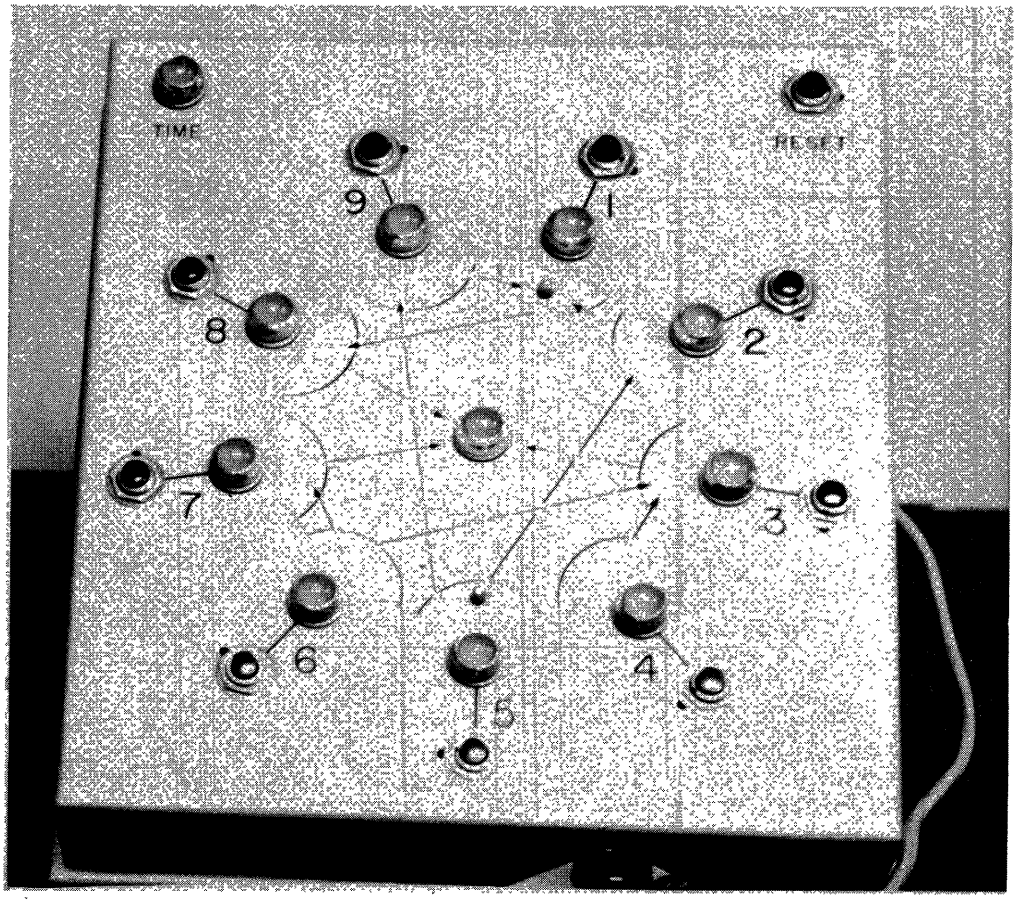


Fig. 1. The Logical Analysis Device Operator's Display Panel

A METHOD OF VOICE COMMUNICATION WITH A DIGITAL COMPUTER

S. R. Petrick and H. M. Willett

Air Force Cambridge Research Laboratories, AFRD
Bedford, Massachusetts

Summary

A pattern recognition procedure for achieving automatic identification of spoken words has been developed and instrumented using an eighteen channel vocoder and a general purpose medium scale computer, the AFCRL Cambridge Computer. The process depends upon the representation of a spoken word by a sequence of octal digits which describe the amount of instantaneous power in each of eighteen frequency bands at time intervals of 1/50 second. Essentially, recognition is achieved by matching such a digital representation of a spoken word against a set of stored word "masks", one for each word to be recognized.

To develop such a set of masks (giving the computer a particular vocabulary) the speaker repeats a word several times. A mask is then computed which optimizes a certain recognition parameter. The speaker must then type into the computer the printed word he wishes associated with his spoken word. This process can then be repeated to add other desired arbitrary words to the computer's vocabulary. At present, using the Cambridge Computer which has only 1600 magnetic drum storage registers, this vocabulary is limited for most purposes to 83 spoken words and requires about one and one half seconds per vocabulary word for recognition.

If the speaker's own voice is used to prepare masks of the words he wishes to be recognized, correct identifications are made with almost 100 percent accuracy. In other cases the degree of success is highly dependent upon the particular individuals involved.

Other programs and procedures which have been tested, all dependent upon the basic word recognition facility, include:

1. The voicing of a word in one language followed by its typewriter association in a second language, resulting in a crude word for word machine translator.

2. A routine which enables a speaker to say a sequence of words (from the set zero, one,..., nine, plus, minus, times, bracket, equals) which are followed by a print out of the words spoken and the value of the expression defined.

3. A speaker recognition program which identifies the talker with appropriate comments as well as the word he spoke.

4. An adaptive program which enables the computer to automatically reorient itself to a new speaker's voice.

Introduction

The recognition of digitalized speech signals has received considerable attention in recent years [1,2,3,4,5,6,7,8,9] for two good reasons. First, many data processing problems would be most conveniently supplied with input of a vocal nature. Examples are plentiful if the vocal input is sufficiently accurate, economical, and general in application, i.e., free from limitations such as individual speaker differences or inability to accept continuous, unsegmented speech. Application is, of course, limited, if the vocal input does not meet the previously specified qualifications, but is not necessarily precluded. This paper will present the capabilities and limitations of one possible procedure for vocal communication with a general purpose digital computer, and it will be left for the reader to supply the applications, if any, of interest to him which seem feasible.

A second explanation of the recent interest in speech pattern recognition is that the current vogue for research in the pattern recognition area of artificial intelligence requires research vehicles. The ability to sort unknown events or objects, each described by a set of numbers, into classes or categories defined only by samples of their known members is basic to machine learning. Spoken word recognition is a particularly good research vehicle in this field because it can provide a wealth of useful data for analysis and testing, because equipment exists for producing digitalized speech, and because there is a large relevant fund of knowledge in the speech field which is of value.

Description of Equipment Used

On the Choice of Input Data

Recognition of any pattern by means of a digital computer, whether arising from a speech source, a printed page, or elsewhere, depends upon finding a digital representation of each event which is to be considered. The digital representation may contain enough information to synthesize an approximation of the original

event or object, or it may merely consist of enough information to insure separation from other members of the population with which we are dealing. If we are interested in doing the best possible job in a particular pattern recognition application, the latter case is to be preferred. Indeed, if we know or can determine which characteristics to measure as basic input data, the subsequent pattern recognition procedure can be made extremely simple. In this case a short truth table is all that is necessary. This approach has been applied, apparently very successfully, to printed character recognition[10], and it is also being considered by several groups for application to speech recognition. The truth table approach, however, requires that the pattern recognition be essentially done by measuring equipment, carefully constructed to solve a particular problem by ingenious human designers. This approach is, therefore, not of primary interest to researchers in pattern recognition. However, for many practical applications the best available measurements of an object we wish to recognize will still require truth tables of prohibitive size and thus depend upon application of more sophisticated pattern recognition procedures. Speech recognition seems to be one of these problems.

The data used in this study were of the previously mentioned, highly redundant variety. This is known because the digit stream of numbers supplied to the computer have been reassembled by a speech synthesizer into intelligible speech. In fact, this equipment about to be described was constructed as a means of reduced bandwidth voice communication with a human listener. The primary reason these data were chosen is that they were readily available. The degree of success of the rather simple pattern recognition method of this paper on such redundant data would seem to indicate a great promise for rapid advancement in oral communication with a digital computer in the near future.

Speech Digitalizing Equipment

The first vocoder was developed about twenty years ago at Bell Telephone Laboratories to investigate improved methods of speech transmission. In recent years, the military services have sponsored considerable reasearch in speech bandwidth compression in order to squeeze more channels into the radio spectrum. The vocoder is basically a device which converts spoken utterances to time - frequency patterns of spectral energy. The particular vocoder used in this study decomposes sound energy into eighteen segments of the audio spectrum using a like number of bandpass filters. [11,12] The output from each of these filters passes to a corresponding spectrum analyzer which measures the power density of the sound in its assigned frequency range. The output from each spectrum analyzer is then fed to an electronic time multiplexer which samples the analyzer outputs at the rate of fifty times per second. Each of these magnitudes is then converted into a three bit binary number. It is thus seen that speech is converted to a sequence of eighteen digit octal numbers at the rate of fifty such numbers or speech "patterns" per second. Figure 1 shows a block diagram of this digitalizing equipment and Figure 2 shows a typical analog speech spectrograph and the corresponding digitalized speech patterns.

Digital Computer Usage

The patterns from the vocoder are read directly into the AFCRL Cambridge Computer through a real time data register. Each fifty-four bit patterh is decomposed into six nine bit units which sequentially enter this ten bit real time input register. The Cambridge Computer is a 1600 word magnetic drum machine, a prototype of the Univac Solid State Computer, and its serial nature barely allows it to store the input data as they arrive with no time remaining for any simultaneous computation. It is only possible to discard initial null patterns and to count the number of patterns to be accepted. The limited storage of the Cambridge Computer permits taking about fourteen seconds of speech if storage is to be completely filled with data or a lesser amount if data is to be processed immediately.

In real time word recognition the speech input has been necessarily limited to two seconds of speech during which an isolated word is to be spoken. Upon storage of this data the computer uses a set of empirical rules to determine the boundaries of the digitalized spoken word, eliminating extraneous background noise and allowing for possible periods of silence within a word. The data are next time normalized so that each spoken word is represented by a fixed number of equally spaced patterns, and the octal digits of these patterns are unpacked and converted to BCD digits. At this point the computer is ready to use this digitalized version of a spoken word for either learning or recognition.

Operating Characteristics Of The System

The word recognition procedure of this paper consists of two modes, learning and recognition. In the learning mode the speaker pushes a button, initiating speech intake by the computer, and speaks a word into the microphone. After this procedure is repeated a predetermined number of times, usually only once more, the computer requests a type-in of the word which has just been spoken. Following this labelling, a mask or template is computed which characterizes this typed word. When this mask is stored in the computer memory, the word it denotes is added to the vocabulary of words which can be recognized. After each mask is computed, the system is ready for the next new word to be spoken.

When operating in the recognition mode with a vocabulary comprised of previously computed word masks, the talker again pushes a button and speaks his word. The computer identifies and types this word, and the system is ready to accept another spoken word. This recognition is accomplished by selecting (using the list of stored masks) that word whose mask most closely resembles the unknown word for which identification is desired.

Word Recognition Method Employed

The previous section used several phrases including "which characterizes this typed word" and "most closely resembles" which must be precisely explained. A number of choices could be made as to how masks should be determined and how they should be matched, and one cannot know a priori which choices are best for the application in question, digitalized word recognition. The particular method used by the authors which will be detailed below is only one of many plausible alternatives. It would admittedly not suffice for many pattern recognition applications, but it has proved useful in dealing with speech.

If we denote by X_{ij} the $i^{\underline{th}}$ spectral component of the j^{th} repetition of some word. and if we denote by Y_i the $i^{\underline{th}}$ spectral component of a stored word mask, the criterion chosen for measuring the agreement between X_{ij} and Y_i is given by

$$C = \left(\sum_{i=1}^{N} X_i Y_i\right) / \sum_{i=1}^{N} (X_i - Y_i)^2$$

If we sum over the various repetitions of that word which are available for mask determining purposes, and if we ask that the function

$$Q = \left(\sum_{i=1}^{N} \sum_{j=1}^{M} Y_i X_{ij}\right) / \sum_{i=1}^{N} \sum_{j=1}^{M} (Y_i - X_{ij})^2$$

be maximized, we can find those mask components Y_i which do this. The details of this maximization will not be exhibited here, but the desired components are given by

$$Y_i^2 = \left(\sum_{j=1}^{M} X_{ij}\right)^2 \left(\sum_{i=1}^{N} \sum_{j=1}^{M} X_{ij}^2\right) / M \sum_{i=1}^{N} \left(\sum_{j=1}^{M} X_{ij}\right)^2$$

This mask is the desired optimal template in the previously defined sense for use with criterion C in effecting a decision procedure.

There are several reasonable decision procedures which could be used. In dealing with a fixed vocabulary one could compute C for each mask using an unknown word and then choose the mask which produced the largest C. Alternatively, one could employ individual thresholds for each mask, or a single threshold for all masks, or perhaps a combination of the above procedures. All of these have been investigated, and some suffice in one application but have disadvantages in others. The procedure used in each application will be included in the discussion of that application.

Word Recognition Conclusions

Results. Having specified the recognition procedure to be used, the next question is, how well does it work? More specifically, how well does it work as a function of those parameters at our disposal? These results were presented orally[13] at the October meeting of the Acoustical Society of America and they are currently being assembled for written publication, so merely the highlights will be given here. Figure 3 shows the effect of varying the time normalization to 18, 9, 5, and 3 patterns per word. The restricted vocabulary here consists of the decimal digits zero through ten, four samples were used in computing each mask, the same speaker's voice was used for both learning and recognition but no utterance was so used for both, nine male speakers are represented, all eighteen frequency channels were retained, and the decision procedure was merely to select the decimal digit whose mask produced the largest value of the previously defined criterion C. The ratio of selection indicated in figure 3 is the average ratio of the values of C for the best and next best words.

Figure 4 shows the effect of channel merging on word recognition. Adjacent channels were averaged to give nine, six, and three frequency channels for each of the time normalizations of nine, five, and three patterns per word. The other specifications as to vocabulary, etc. are the same as already given for figure 3.

Figure 5 shows four typical confusion matrices illustrating recognition of phonetically similar words. The number n in a particular row and column implies that the word in that column was spoken and identified as the word in that row n times. One speaker was used throughout and otherwise the specifications of the previous two figures apply. Similar matrices exist for a number of other frequency-time normalization combinations and their principal results are displayed in figure 6.

Summarizing briefly, very few recognition errors are made if enough bits are retained and if a single speaker is used both for mask making and for subsequent recognition. If, however, different individuals are used for learning and recognition, the results are highly dependent upon the individuals in question. Performance varies from nearly perfect to consistently in error for certain people and words. One solution to this difficulty is to make up composite masks from several good speakers. This has been found

to increase performance but is not easily mechanically accomplished since a speaker may say certain words fine for the purpose of universal recognition but say others quite ambiguously for this purpose. Another solution is the adaptive program to be described in the next section. It will be seen, however, that this procedure has certain limitations, and in these cases the fastest way to insure good performance for a particular speaker's voice is to make masks from his voice. Fortunately, this is easily accomplished for an arbitrary vocabulary by speaking directly those words whose recognition is wanted.

Limitations. In addition to the previously enumerated accuracy limitations there are other restrictions of the method of this paper which should be carefully stated. One of these concerns storage requirements. Using five patterns per word and nine frequency channels, six Cambridge Computer words of storage were reserved for every desired vocabulary word. Because of the Computer's limited storage, only 500 storage registers are available for mask storage, limiting the computer vocabulary to 83 such vocabulary entries. The remaining Cambridge Computer storage is used as follows: 200 registers for the recognition program, 200 words of temporary storage for the unknown input word and 100 registers for its time and frequency normalization, 200 storage registers for the real time data input program, 200 registers for decoding the scrambled input bits and converting them to BCD characters, 100 registers for elimination of coughs and background noise, and 100 registers for storage of alphabetic responses.

Of course, if more storage were available, input durations of more than two seconds would be feasible, assuming real time recognition were not possible on a word at a time basis. Words must still, however, be spoken in isolation or else continuous speech must be segmented into words. Work on the segmentation problem is presently under consideration by various groups.[14]

A final limitation that will be mentioned is computation time required. This depends, of course, upon the number of masks in storage. Using the rather slow drum computer of this study about one and one half seconds per stored vocabulary word (5 patterns - 9 channels) are required. This figure would be cut by a factor of about 100 if a computer of IBM 704 speed were used. If we are satisfied with successive words occurring no faster than every n seconds, we could then achieve real time recognition on a word at a time continuous basis for a vocabulary of about $66n$ words.

Demonstration Programs

Basic Word Recognition Program. It has been seen that the basic recognition computer program previously described allows one or more speakers to build up a working arbitrary vocabulary with a minimum of time and effort merely by speaking each word into a microphone several times. Initially, the computer requests the user to type the number of patterns per word he wants for time normalization and to also type the number of word repetitions to be used for the computation of each mask. When this information has been supplied, the first word may be spoken. Processing of each repetition of a word takes about three seconds. Following the computer's request for alphabetic labelling of a spoken word, the mask and corresponding alphabetics are punched out on paper tape. The system is then ready for the next word to be spoken. When masks have been made for all desired vocabulary words, recognition can proceed for those words. The limited storage of the Cambridge Computer necessitated separate programs for mask making and subsequent recognition. Accordingly, to proceed, the recognition program and desired vocabulary must be read into the computer. Channel merging, if desired, can be accomplished while the data are being entered into storage.

The decision procedure used for this demonstration program selects a word immediately provided the value of C its masks produces is higher than a given absolute threshold. If no decision is made in this manner, the word whose mask produced the highest C is selected providing this value is greater than a lower threshold. If the best choice is below this minimal threshold, the computer requests that the word be repeated again. The thresholds were empirically selected, and values were found which produced very few requests for word repetitions and virtually no false identifications.

Language Translation Program. In order to effect a crude word for word translator from some spoken language to a different written language it is obviously only necessary to change the alphabetics used for displaying words recognized. This was done as a demonstration for the authors' Laboratory Chief using the German decimal digits null through zehn as spoken input and English numerals for printed response. Sufficient similarity was found between the German pronunciation of the demonstratee and demonstrator, whose voice was used to produce the masks, to permit translation for both without error. This exercise was, of course, limited to a relatively small sample size, and while it may not prove much about the effectiveness of the word recognition procedure of this paper, it would seem at least to indicate that the author in question's German pronunciation couldn't be too bad.

Speaker Recognition Program. It was observed that the indices C obtained were considerably higher when the same individual was used for both mask computation and subsequent recognizing. This was exploited to effect speaker recognition of individuals speaking from a restricted vocabulary. In one exercise nine male and seven female voices were used to produce masks for the words "one", "two", and "three". Nine frequency channels, five time samples per word, and four word repetitions were used for each of the forty-eight

masks. Different word repetitions were utilized for the recognition phase. Each of the sixteen speakers spoke "one", "two", and "three" several times, and the choice of speaker and digit spoken was made solely by selecting the largest value of the index C. Figure 7 shows the results of this study. The numbers in each box indicate the sample size and percent of successful identification.

Encouraged by these results, a speaker recognition program was written for demonstration purposes, using the previous forty-eight word masks. With this program an unknown speaker says his word into the computer and obtains one of the following types of responses: "That was John Jones speaking the digit three; I don't know who you are, stranger, but you spoke the digit two; I don't have the slightest idea who you are or what you said; speak more distinctly and repeat your word again, please." The remaining possibility, "I don't know what you said, John Jones, but I recognize you", is not currently allowable but is under active consideration, and it appears to be feasible with at least a fair degree of success.

The decision procedure used for the above program is the following. Each of the categories involving both a specific speaker and spoken word are selected only if their associated value of the criterion C exceeds a threshold of the form A+ BQ where A and B are constants and Q is a number associated with each mask and computed at the same time as the mask. This is actually the same maximal Q which was previously defined. It can be shown that the maximal value of Q is given by

$$Q = 1/\ 2(K\sqrt{M} - 1)$$

$$\text{where } K^2 = \sum_{i=1}^{N}\sum_{j=1}^{M} X_{ij}^2 \Big/ \sum_{i=1}^{N}\Big(\sum_{j=1}^{M} X_{ij}\Big)^2$$

and this expression is the one which was used to compute Q. The primary reason for using Q is empirical, but it was suggested by the following qualitative thinking: Q is of the form

$$(n_1 + n_2 + \ldots + n_M)/(d_1 + d_2 + \ldots + d_M)$$

and if each of the M word repetitions in the sample are sufficiently similar, each n_j and d_j will not differ excessively from

$$\bar{n} = \Big(\sum_{j=1}^{M} n_j\Big)/M$$

and $\bar{d} = \Big(\sum_{j=1}^{M} d_j\Big)/M$.

Under these assumptions we can approximate Q by $\bar{n}\ M/\bar{d}\ M$ or $\bar{n}/\bar{d}$.

If we now compute a value of the criterion C $= n/d$ from the mask which maximizes Q, we should find that C approximates Q because $n/d \approx \bar{n}/\bar{d}$ if n and d are close in value to $\bar{n}$ and $\bar{d}$. This reasoning prompts us to consider BQ, $0 < B < 1$ as a threshold against which C may be compared. The other constant A was added to BQ strictly for empirical reasons.

If index C does not exceed any threshold A + BQ, the best choice of one, two, or three is made for each speaker, and if at least p of the speakers agree on the same choice, this digit is selected and the speaker is assumed to be someone not represented by a stored mask. If less than p speakers agree upon the digit spoken it is assumed that a word other than one, two, or three was spoken. Finally, word repetition is requested only when some probable source of trouble is detected in the input program such as improbable length of the spoken word.

Scope Display Program. One of the distinctive output features of the Cambridge Computer is a large 19 inch three color (red, blue, green, and combinations thereof) scope display unit. This was used to allow easy visual scanning of spectral word representations. The horizontal axis is the time axis and the vertical axis is used for frequency. At any intersection the color of the point denotes magnitude of spectral energy with several color coding choices available. Input may be either from punched paper tape or real time from the microphone. This program was found to be of value not only for quickly insuring that all equipment involved is working all right initially, but also for diagnosing excessive noise and obtaining useful information about the structure of certain digital word representations. For example, in examining the similar words bit and beat for one speaker it was discovered that he inserted a prominant period of silence before the final stop consonant in the word beat but did not do so for the word bit. This indicates machine separation of those two words for that speaker was based on more than the phonetic vowel difference between $\breve{e}$ and $\breve{i}$

Arithmetic Expression Evaluation Program. This program utilizes a vocabulary consisting of the decimal digits zero through nine and the operation symbols plus, minus, times, bracket,and equals. The decision procedure used is the same as that for the previously described basic word recognition demonstration program.

In using this program the speaker makes up a meaningful arithmetic expression from the allowable input words. Each word is sequentially recognized and symbolically stored until the word "equals" is encountered. At this point the expression is evaluated, multiplication first and then addition and subtraction in cases not specified with brackets. The entire expression understood by the computer is then typed followed by its computed value. Examples run on the

computer include:

2+2 = 4

[13-5]*6 = 48

123+456*3-2*[26-23]+35*[165-166]-4 = 1446

314*[1327-64*21]*[129*4+62*72*2-89*43] +[43*2196]-17-[26-24*3+4]*[7-9] = -29889219

Adaptive Reorientation Program. While the method of this paper was found to be very successful when the same speaker's voice was used for both mask computation and word recognition, results are less consistently successful when different voices are involved. To eliminate this difficulty, two adaptive programs were written, both designed to convert a set of masks made from one speaker's voice to a set corresponding more closely to a new speaker. One of these programs requires the use of an operator to make certain decisions and the other program operates independently of any human intervention. In both cases the masks of words taken from one or a group of speakers constitute the computer's vocabulary. An unknown speaker then says a word which may or may not be represented in the vocabulary.

In the first case, the ratio of the highest value of the agreement index C for this word to the next highest C is taken, and this value, R, is compared against a set of thresholds. If R exceeds the highest threshold. the computer prints the recognized word and awaits the next word. If R fails the highest but exceeds the next lower threshold, the computer prints its choice and asks the operator to indicate if this choice is right or wrong. If the correct choice is made, a fraction f_1 of the word is averaged into the mask for that word. If, however, the wrong choice is made, the operator types the actual word spoken into the machine. The computer then scans its vocabulary to see if that word is represented. If the word is already on the vocabulary list, a larger fraction f_2 , is averaged into the mask. If, however, the word is not found. it is added to the end of the vocabulary, thus becoming the mask representation of that word.

If neither threshold is exceeded, the computer asks the operator to identify the word. Again the vocabulary list is scanned and the proper mask, if found, is adjusted by a still larger fraction f_3 . Also, if the word is not found, it is added to the vocabulary.

If the mask has been adjusted (i.e.,in any case except the first where the highest threshold has been exceeded), the computer asks that the word be repeated. If a word is missed more than once, the fraction of the word which is averaged into the mask is increased. After n misses, the computer either stops or, at the option of the user, discards its earlier attempts and replaces the mask by the last repetition of that word.

The second learning program, as mentioned, operates without human intervention. Again the ratio R is computed from the two highest values of the agreement index C. Similarly, if R exceeds this program's highest threshold, no adjustment is made. However,if R fails to exceed the lower threshold, the word is rejected and no adjustment is made. In any other case, the fraction of the word to be averaged into the masks is determined from the value of R obtained. This fraction is taken to be 3/4 when R equals the lower threshold and decreases linearly as a function of R to zero when R equals the higher threshold.

The results obtained from running both adaptive programs seem to indicate that the programs are of practical value as a means of obtaining new masks only when the initial agreement is substantial. With operator intervention, of course, successful conversion is always achieved. but if a minimum of human participation is required, the making of new masks from scratch is to be preferred. The other adaptive program may or may not converge successfully to the new speaker's voice. One approach which might improve the performance of both programs involves the previously mentioned use of composite masks made from several good speakers.

As an example of a typical attempt to convert from one speaker's (WD) voice to another's (AP) and back without operator assistance the following run is presented. A mask for the single word "one", "two", and "three" spoken by WD. The procedure here being followed using taped input words is to repeat the identical utterance if R lies between the two thresholds. The two values chosen as thresholds here were two and ten.

Step	Speaker	Utterance	R
1	AP	5	1.63
2	AP	6	1.84
3	AP	7	2.31
4	AP	7	> 10
5	AP	8	3.21
6	AP	8	> 10
7	WD	6	1.8
8	WD	7	1.65
9	WD	8	1.5
10	WD	9	1.99
11	WD	10	1.54
12	AP	5	6.0
13	AP	5	>10
14	AP	6	5.72
15	AP	6	>10
16	AP	7	>10
17	AP	8	>10

In steps 1 and 2 AP's utterances did not produce enough agreement with WD's "one" to exceed the threshold two. Accordingly no action was taken. The next utterance (utterance seven in step three), however, gave R = 2.31 causing the first mask modification to be effected. Repeating this same utterance now gave a value of R in excess of the upper threshold (step 4). Step 5 again modifies the mask. In steps 7 through 11 WD is speaking "one", but recognition is too poor to cause any mask modification in his favor. In steps 12 through 17 AP's utterances are again repeated until no more modification is made.

Conclusions

The method of this paper seems feasible for those applications where words are spoken in isolation by a speaker whose voice was used to prepare masks. These masks may be made rather easily and retained for each talker who wishes to be able to communicate orally with the computer. The limited storage capacity of the Cambridge Computer restricted the size of the vocabulary possible at any one time. However, the observed discrimination between phonetically similar words seems to indicate that words could be recognized from a larger vocabulary. Use of a large scale computer and a 500 word vocabulary is presently under consideration.

Acknowledgments

The authors are grateful to C. P. Smith and his group who developed the vocoder system employed and operated this equipment whenever speech data was taken. The scope display program and real time input program were written by F. H. Cook, who also assisted in designing input equipment necessary to read in real time data.

References

1. K.H. Davis, R. Biddulph, and S. Balashek, "Automatic Recognition of Spoken Digits," Journal of the Acoustical Society of America, Vol. 24 (1952), pp. 637-642 and Communication Theory (Butterworths Scientific Publications, London, 1953), pp. 433-441.

2. J. W. Forgie and G. W. Hughes, " A Real-Time Speech Input System for a Digital Computer," Journal of the Acoustical Socieity of America, Vol. 30 (1958), p .668.

3. D. B. Fry and P. Denes, "Mechanical Speech Recognition," Communication Theory (Butterworths Scientific Publications, London, 1953), pp. 426-432.

4. D. B. Fry and P. Denes, "Experiments in Mechanical Speech Recognition," Information Theory (Butterworths Scientific Publications, London, 1956), pp. 206-212.

5. D. B. Fry and P. Denes, "On Presenting the Output of a Mechanical Speech Recognizer," Journal of the Acoustical Society of America, V l. 29 (1957), pp. 364-367.

6. G. W. Hughes and M. Halle, "On the Recognition of Speech by Machine," Proceedings of the June 1959 International Conference on Information Processing, Paris, (Butterworths Scientific Publications, London, 1960).

7. M. V. Mathews and P. Denes, "Spoken Digit Recognition Using Time-Frequency Pattern Matching," Journal of the Acoustical Society of America, Vol. 32 (1960), p. 914.

8. H. F. Olson and H. Belar, "A Phonetic Typewriter," Journal of the Acoustical Society of America, Vol. 28 (1956) pp. 1072-1081.

9. G. Sebestyen, "Automatic Recognition of Spoken Numerals," to be abstracted in Journal of the Acoustical Society of America, V l. 32 (1960).

10. "The Voracious Eye", Time Magazine, Vol. LXXVI No. 10, Sept. 5, 1960, pp. 69-70.

11. C. P. Smith, "Speech Data Reduction," AFCRC-TR-57-111, ASTIA Document No. AD117290, May 1957.

12. C. P. Smith, "An Approach to Speech Bandwidth Compression," Proceedings of Seminar on Speech Compression and Processing, AFCRC-TR-59-198, Vol. 2, Sept. 1959.

13. S. R Petrick and H. M. Willett, " Digital Automatic Word Recognition Procedure," to be abstracted in Journal of the Acoustical Society of America, Vol. 32 (1960).

14. J. W. Forgie and C. Forgie, "Segmentation Scheme for Use of a Speech Recognition Computer Program," to be abstracted in Journal of the Acoustical Society of America, Vol. 32 (1960).

CONVERSION OF VOICE SPECTRUM PATTERNS TO DIGITAL WORDS

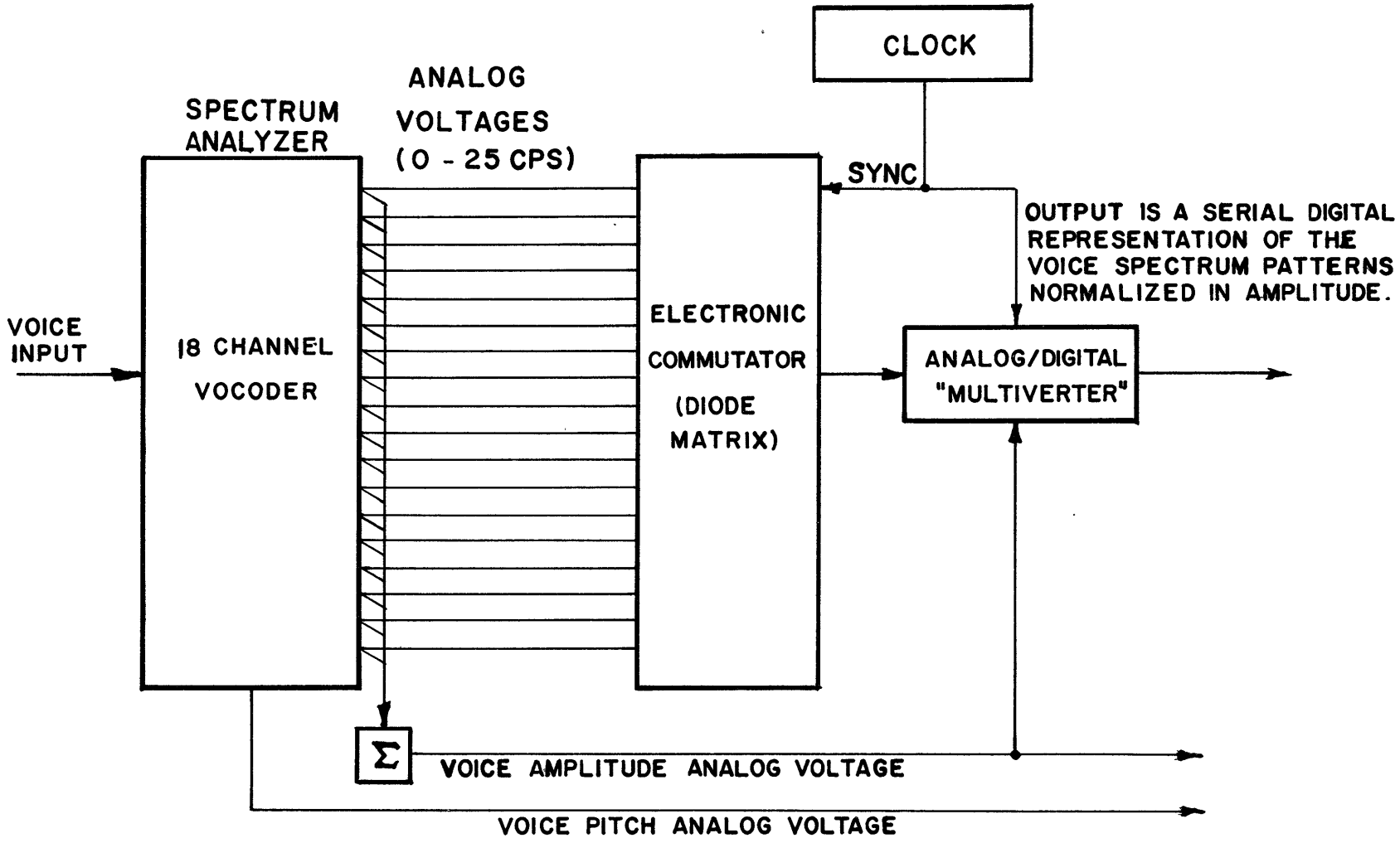


Fig. 1.

FREQUENCY
0
1
2
3
4
5
5 KCPS
FREQUENCY
5 KCPS
3 - ⑤ 24 APRIL 59 CPS
END OF TEST NUMBER THREE
TIME
100 MSEC

Fig. 2.

WORD RECOGNITION OF DECIMAL DIGITS

EFFECT OF TIME NORMALIZATION

NUMBER OF PATTERNS PER WORD	SAMPLE SIZE	% SUCCESS	RATIO OF SELECTION
18	218	99.5	6.4
9	97	100	6.9
5	104	99.2	10.3
3	43	95	4.5

Fig. 3.

WORD RECOGNITION OF DECIMAL DIGITS

EFFECT OF CHANNEL MERGING

ON WORD RECOGNITION

NUMBER OF PATTERNS PER WORD	NUMBER OF FREQUENCY CHANNELS	SAMPLE SIZE	% SUCCESS	RATIO OF SELECTION
9	9	77	98.7	10.14
9	6	77	96	7.6
9	3	76	96.1	5.1
5	9	77	100	11.1
5	6	77	94.8	9.1
5	3	75	93.5	5.4
3	9	77	94.7	10.0
3	6	77	93.5	6.8
3	3	78	86	∞

Fig. 4.

CONFUSION MATRICES ILLUSTRATING RECOGNITION OF PHONETICALLY SIMILAR WORDS

SPOKEN
5 TIME SAMPLES
18 FREQUENCY CHANNELS

	BAT	BATE	BEAT	BET	BIT	BITE	BOAT	BOOT	BOUGHT	BOUT	BUT
BAT	4										
BATE		4									
BEAT			4								
BET				4							
BIT					4						
BITE						4					
BOAT							4				
BOOT								4			
BOUGHT									4		
BOUT										4	
BUT											4

SUCCESS RATE 100 %

SPOKEN
3 TIME SAMPLES
18 FREQUENCY CHANNELS

	BAT	BATE	BEAT	BET	BIT	B.	BOAT	BOOT	BOUGHT	BOUT	BUT
BAT	4										
BATE		4									
BEAT			4								
BET				4							
BIT					4						
BITE						3					
BOAT							4	1			
BOOT								3			
BOUGHT									4		
BOUT										4	
BUT						1					4

SUCCESS RATE 95.45 %

SPOKEN
3 TIME SAMPLES
6 FREQUENCY CHANNELS

	BAT	BATE	BEAT	BET	BIT	BITE	BOAT	BOOT	BOUGHT	BOUT	BUT
BAT	4			1							
BATE		1			1						
BEAT		3	4								
BET				3	1						
BIT					2						
BITE						1					
BOAT							4	3			
BOOT								1			
BOUGHT									3		
BOUT									1	4	
BUT						3					3

SUCCESS RATE 68.18 %

SPOKEN
3 TIME SAMPLES
3 FREQUENCY CHANNELS

	BAT	BATE	BEAT	BET	BIT	BITE	BOAT	BOOT	BOUGHT	BOUT	BUT
BAT	2										
BATE	1										
BEAT											
BET		3		2	1					2	
BIT		1	3								1
BITE	1					1					1
BOAT									1	1	1
BOOT								4			
BOUGHT			1		1		3		3	1	1
BOUT				1	1		1				
BUT				1	1	3					

SUCCESS RATE 27.27 %

Fig. 5

RECOGNITION OF PHONETICALLY SIMILAR WORDS AS A FUNCTION OF THE NUMBER OF FREQUENCY CHANNELS AND TIME SAMPLES.

NUMBER OF FREQ. CHANNELS	NUMBER OF TIME SAMPLES PER WORD	TOTAL NUMBER OF DIGITS	SUCCESS %
18	3	54	95.5
9	3	27	86.7
6	3	18	68.3
3	3	9	27.3
18	5	90	100
9	5	45	86.5
6	5	30	77.8
3	5	15	54.6
18	9	162	100
9	9	81	100
6	9	54	95.5
3	9	27	80
18	18	324	100
9	18	162	100
6	18	108	93.2
3	18	54	78

Fig. 6

SPEAKER IDENTIFICATION

	MEN	WOMEN	BOTH
1	54 94 %	39 100 %	93 97 %
2	54 96 %	39 100 %	93 98 %
3	54 89 %	43 100 %	97 94 %
TOTAL	162 93%	121 100 %	283 96%

9 MALE SPEAKERS
7 FEMALE SPEAKERS

Fig. 7

FILTER - A TOPOLOGICAL PATTERN SEPARATION COMPUTER PROGRAM

Daphne J. Innes
Lawrence Radiation Laboratory
Berkeley, California

Summary. The advent of high energy particle accelerators and liquid bubble chamber detectors has added the demands of high speed data reduction to the many problems of modern nuclear physics research. For example, one six-month experiment on the University of California 72-inch Hydrogen Bubble Chamber yields photographic records of millions of nuclear events. This paper discusses one of the new measuring and topological identification devices which has been developed to analyze these great volumes of research data.

Dr. Bruce McCormick has proposed a scanning technique which allows rapid recognition, separation and measurement of the photographic records of star type nuclear events. A device known as the Spiral Reader measures background and star type event features impartially, discriminating against non-radial patterns by the geometry of its rotating scanning element. The event measurements are separated from the background measurements by an IBM 704 computer under the direction of a program called FILTER. The separated nuclear event measurements are subsequently reconstructed in space for physics analysis.

FILTER exploits the observation that if a segment of a circular arc is rotated about a point on that arc, intercepts occur at regular intervals along a radius to the point at constant angular intervals of the rotation azimuth. The Spiral Reader, by placing the burden of event discrimination on a high speed digital computer, minimizes the need for either special analysis equipment or for a human operator to make the topological separation. Simulation, calibration and cathode ray tube display routines have been included in the filter system.

This paper describes the computer program FILTER which separates the topological star-type event configurations from undesired background features.

Many modern high energy nuclear physics experiments are performed by observing the interactions of elementary particles in a liquid Bubble Chamber. Tracks of tiny bubbles in the liquid define the paths of ionized particles, much as do the fog droplets in the familiar Wilson Cloud Chamber. Stereo photographs of the chamber preserve these tracks so that the nuclear interactions may be analyzed. A typical 72-inch Hydrogen Bubble Chamber experiment at the Lawrence Radiation Laboratory produces six stereo triad photographs every minute. As almost every triad includes a nuclear interaction, several million events are available for analysis each year.

A stereo triad of the 72-inch chamber comprises three views, each 33mm by 125 mm. An average bubble image diameter is 40 microns on the film. The track images are opaque against a clear background, the film being a negative of the dark field illuminated chamber. Figure 1 is a typical photograph of the chamber.

Human operators search the films for interesting nuclear interactions, designating those track combinations which are to be measured. Rectangular coordinate point measurements along the participating event tracks define the track locations with respect to reference marks in the chamber. The measuring technician must select the event tracks from the numerous background features as he directs the operation of the currently used semi-automatic measuring projector[1]. An IBM 704 computer program, PANG, makes a spatial geometrical reconstruction of the event from the measured two-dimensional data[2,3].

The topologically reconstructed event is kinematically fitted to physics hypotheses by a second computer program, KICK[5]. The physicist makes experimental conclusions and evaluations from these analyses.

One of the most numerous topological event types is the single vertex interaction. Figure 2. The currently available measuring techniques, being completely saturated by processing the rarer interactions, must neglect the important research information of these events. Dr. Bruce McCormick has proposed a scanning method especially suited for measuring such star-type events. A magnified image of the event is projected on a rotating disk. The event vertex image and the center of the rotating disk are superimposed. This disk is opaque except for one segmented radial slit. The slit is about one-third as wide as is the image of a typical track projected upon it. Each of the radial segments scans an annular path of ten average track widths long. Figure 3 indicates the annular areas scanned around a vertex. This scanning technique discriminates geometrically against tracks which do not emanate from the event vertex. A photomultiplier collects light passing through the scanning slit. As the disk rotates over the image, the electrical signal from this photomultiplier displays the film density variations. Radial tracks appear as pulses. A rotary analog-to-digital shaft encoder attached to the scanning disk permits azimuthal measurement to be made of the track or feature pulses.

Track pulses which exceed a preset amplitude are squared by a discriminator. The angles at which the leading and trailing edges of these squared pulses occur are the azimuthal pulse-pair measurements of the tracks. A magnetic core buffer memory temporarily stores these

azimuths along with the relative pulse amplitude number and the slit radius. These measurements, a rectangular coordinate vertex location and event descriptive information, are recorded on IBM compatable magnetic tape. This tape is the input data for the event separation program, FILTER.

Bubble chamber photographs are cluttered by non-interacting beam tracks, frost on the optical surfaces, gas bubbles, electron spirals and other irrelevant features. The Spiral Reader makes only a partial attempt to discriminate against these noise features by favoring radially disposed tracks. The burden of separating the event measurements from the background information is placed upon a digital computer program rather than upon human judgement or special complicated equipment.

An effective event separation program must detect all valid tracks in each view. It is permissable for an occasional extraneous track to be mistaken as valid in one of the three views. An erroneous labeling of a background feature can be corrected by comparison with the other two views.

In addition to removing noise measurements from the data, the event separation program must solve several topoligical conditions:

1. Tracks passing through obscuring features are not to be lost.
2. Tracks which branch into two or more tracks are to be separated and identified.
3. Tracks which cross over one another are to be individually defined.
4. Missing data from an occasional gap in the track image is not to stop track separation.
5. The program is to tolerate data distortions from small misalignments of the scanning disk center and the event image vertex.

The radial scanning technique affords advantages other than the obvious discrimination against non-radial tracks. As the rotating scanning disk is centered over the event vertex, all tracks in the event are in principle scanned by the first three inner slits. Actually many event tracks appear in only two of the first three slits because of centering errors and track gaps. The tracks near the vertex appear as straight lines. Therefore all tracking of event tracks may be initiated by finding all pulse combinations in the first three slits which form radial lines. As the area scanned by the first three slits is small, only a few track pulses must be considered for the track initiation part of the program. Typically, ten to twenty pulses are found in the third slit data. Very little time is spent by the computer while making an exhaustive search of the data to form straight line groups. Some of these initial straight line combinations do not represent tracks but are formed by dirt and isolated bubble images. Subsequent program operations remove these anomalies.

No feature which obscures less than one-fifth of the area of a scanning slit is measured. This discrimination often avoids recognizing small particles of dirt and individual bubbles. Tracks which cross a scanning slit at an angle greater than thirty degrees are usually not measured. Unfortunately, short radius tracks are lost by the outer disk slits.

The FILTER program flow diagram is shown in Figure 4. The polar coordinate measurements made by the Spiral Reader are read from magnetic tape into the computer memory. The Gray coded azimuth numbers are converted into binary numbers. Possible malfunctions of the Spiral Reader azimuth digitizing system are detected by verifying that the successive angle measurements from each slit are in ascending order. The measurements must be arranged in sequence from the innermost to the outermost slit.

The azimuth data from each slit are separated into leading and trailing edge pulse-pair combinations, these pulses being stored in memory sections called slit banks. The relative pulse height designating numbers are also stored in the slit banks to be used in a subsequent track ambiguity resolving operation. The pulse-pair words in the slit banks can be regarded as forming a digital pulse train in which each word defines one pulse.

After data storage and verification, the program searches slits 1, 2, and 3 for pulses which form straight radial lines. The straight line fit used to initiate event tracks is not sufficiently good for track interpolation beyond the third slit. We have found that, for the most curved tracks which can be seen by the Spiral Scanning Disk, a lineaar spiral approximates the actual curve with sufficient accuracy to make prediction of a track pulse azimuth as far as two slits beyond the last established pulse on the track to within 50 microns of its true position.

Tracks are reconstructed from the measured data slit-by-slit from the vertex outward. An azimuth is predicted for the next outward slit track pulse by extrapolating a least-squares fit of the already established track pulses. A search zone 100 microns wide is established about the predicted azimuth. The slit bank is entered and a search made for track pulses lying completely or partly within the search zone. The search is effected by comparing the predicted azimuth range with the smallest angle pulse-pair in the bank. If these azimuth ranges do not overlap and the predicted azimuth is greater than this smallest angle pulse, a comparison of the predicted azimuth with the largest pulse-pair angle is made. If again the azimuth zones do not overlap, but the predicted angle lies within the range of the smallest and largest angle pulse-pairs, the predicted azimuth is compared to the azimuth of a pulse-pair at the middle of the list of pulse-pairs for that slit. If again there is no coincidence of azimuths, the number of pulse-pairs to be searched has been reduced to one-half because the predicted azimuth is either greater than or less than that of the middle pulse-pair. This comparison and subsequent halving of the azimuth ranges is continued until either a pulse-pair is found containing the

predicted azimuth zone or the search is abandoned. The maximum number of searches made in one slit bank is equal to the logarithm to the base 2 of the number of pulse-pairs in the bank. Even a slit bank containing 100 pulse-pairs, such as has been found on the 15th slit of the Spiral Reader while measuring noisy film, needs only 7 or 8 searches to define a track.

Since tracks may branch or cross over one another, it is probable that more than one track pulse will be discovered within a search zone. The pulse-pair with the lowest azimuth is tentatively selected as the valid track pulse, the higher azimuth pulses being stored for later investigation.

Figure 5 is a schematic of the pulse trains of the innermost 5 slits in the neighborhood of the branching track shown in Figure 2. Only one pulse appears in each of the search zones for slits 1 and 2, but there are two pulses, A and B, within the slit 3 search zone. The search zone for the slit 4 pulses is predicted by extrapolating the pulses on slits 1 and 2 and 3A. Pulse C is found in the slit 4 search zone.

The search zone for slit 5 is now predicted by fitting the pulses 1-2-3A-4C to a linear spiral. A pulse is discovered within the slit 5 search zone and the tracking is continued to the outermost slits.

When the track of this example terminates, the program returns to slit 3 to test the alternative track established by the presence of pulse-pair 3B. A new search zone is predicted in slit 4 by the linear spiral fit to the points 1-2-3B. In this way, the program finds the two prongs of a branching track. This method of tracking in addition to detecting branching tracks can also generate false tracks-pseudo-prongs.

It is interesting to consider the effects of increasing the search zone from 2 track widths to 4 track widths. See Figure 6. As before, pulse A and B are found in slit 3 but when 1-2-3A is extrapolated, the new search zone on slit 4 includes C and D. Similiarly the search zone predicted by fitting 1-2-3B also includes C and D. We find that the search zones in slit 5 are predicted by 4 pseudo-prongs: 1-2-3A-4C; 1-2-3A-4D; 1-2-3B-4C; 1-2-3B-4D. Clearly, if the size of the search zone is not limited, the number of speudo-prongs generated will rapidly become excessive. The zones must be sufficiently wide to avoid losing true tracks. A search zone 100 microns wide appears to be a good compromise for the 72-inch chamber film.

Figure 7 is employed to describe the effect of dirt or an isolated bubble feature near a true track image. The figure represents the pulse trains on the inner 5 slits of track 3 in the event shown in Figure 2. With a search zone of 100 microns, pulse 3A is separated from pulse 3B and the predicted slit 4 search zone includes no pulse. Neither will any pulses be found in the slit 5 search zone predicted from 1-2-3A; hence the search is abandoned. A noise pulse 4C is so close to 4D that the search zone predicted from 1-2-3B-4C in slit 5 includes a pulse which actually belongs to the track 1-2-3B-4D. The two track candidates generated by the presence of the noise pulse 4C ard the real pulse 4D are stored in track banks for ultimate separation by the program.

The FILTER program is capable of following tracks through large obscuring noise features. Figure 2 shows a thermocouple partially obscuring a track. The thermocouple measurements appear as very wide pulses -- these the program neglects. The extrapolation is continued through the noise, for as many slits as is necessary. During the extrapolation through gaps or noise, a constant angular sector search zone is searched from slit to slit; the search zone is therefore widened as the radius increases.

Often track candidates share pulses with other track candidates. Whether this sharing is an indication that real tracks are branching or crossing or whether the tracks have been created by noise must be determined. Any track containing four or more pulses is tentatively assumed valid. A chi-square test is computed on the geometric fit of the pulse-pairs to a linear spiral for each track candidate. An ionization consistency rating is based upon the fact that the product of the pulse amplitude and pulse width should remain constant along the length of the track. A quality criterion of the track, the weighted sum of these two designators, is stored in the track bank with the pulse measurements.

It is convenient to group the track candidates into sub-sets. A sub-set contains track candidates which intersect any other track candidates in that sub-set. No track candidate in one sub-set intersects any track candidate in another sub-set. The track candidate with the lowest-best quality criterion in each sub-set is selected as a real track. Each other track candidate in the sub-set, in order of increasing quality function, is then compared to the real track. The assumption is made that if any track candidate has two or more consecutive pulses independent from those of any real track and if it has a quality criterion better than a predetermined limit, it too is a real track. Once a track candidate is classified as real, subsequent prong comparisons are made to it as well as to the original track.

This operation has proved adequate to insure that all real tracks are discovered. Noise features close to real pulses on two or more consecutive slits create pseudo-prongs which are difficult to distinguish from close crossing tracks. More stringent criteria would cause the loss of real tracks. After the searching and testing on an individual view basis, a stereo reconstruction of the event is attempted. Since the three stereo views are available for comparison, it has been usually possible to resolve these ambiguities by eliminating tracks which do not appear in at least two of the three views.

For each track, measurements from two views are selected which give the best stereo reconstruction. These measurements constitute the

input data for the subsequent geometric reconstruction program PANG[2]. One view of a single vertex event is typically separated from the background in 3 to 5 seconds of IBM 704 operation.

The ability of the program to separate an event from the irrelevant features on the film is demonstrated by a comparison of the plot of input data to a plot of the reconstructed event data. Figures 8 and 9 are on-line plots of the data made by an IBM 709 from the event shown in Figure 2. These on-line plots are made by the IBM cathode-ray oscilloscope as the input data is processed. This part of the FILTER program, while most useful for the equipment and program development, will not necessarily be used for the production program.

Now that the engineering parameters have been defined by a prototype Spiral Reader and the effectiveness of the FILTER program proved, the Lawrence Radiation Laboratory is constructing a fast scanning and measuring Spiral Reader. This measuring system is expected to process 200,000 single vertex events each year[4]. It is believed that the tracking techniques developed for FILTER can be employed in the realization of a fully automatic bubble chamber data processing system.

Work on this project has been performed in the Lawrence Radiation Laboratory Hydrogen Bubble Chamber Physics Group under the direction of Dr. Luis W. Alvarez. The Spiral Reader project was originated by Dr. Bruce McCormick, now at the University of Illinois.

This work has been supported by the United States Atomic Energy Commission.

References

(1) A. H. Rosenfeld, International Conference on High Energy Accelerators and Instrumentation, CERN 1959 "Digital Computer Analysis of Data from Hydrogen Bubble Chambers at Berkeley."

(2) W. Humphrey, A Description of the PANG Program, Alvarez Physics Memo 111, 115.

(3) A. Grasselli, B. McCormick, J.A.G. Russell, A Comprehensive Data Analysis System for the 72-Inch Hydrogen Bubble Chamber, submitted to the Atomic Energy Commission Division of Research, November 23, 1959.

(4) J. A. G. Russell, The Spiral Reader -- Phase Two, Engineering Note S.R. 19, March 1960.

(5) A. H. Rosenfeld, A Description of the KICK Program, UCRL 9098

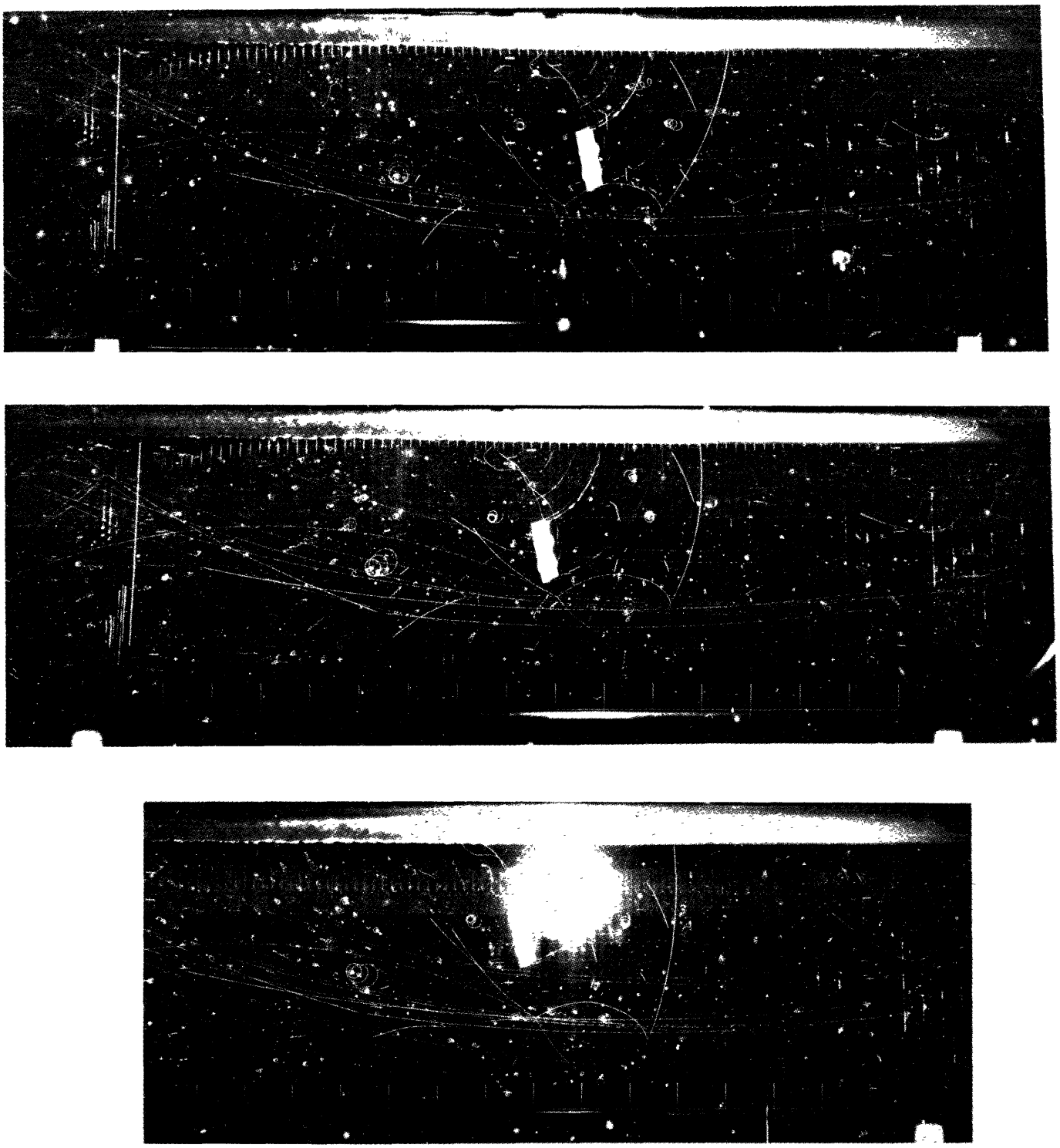


Fig. 1. A typical 72-inch Hydrogen Bubble Chamber stereo triad

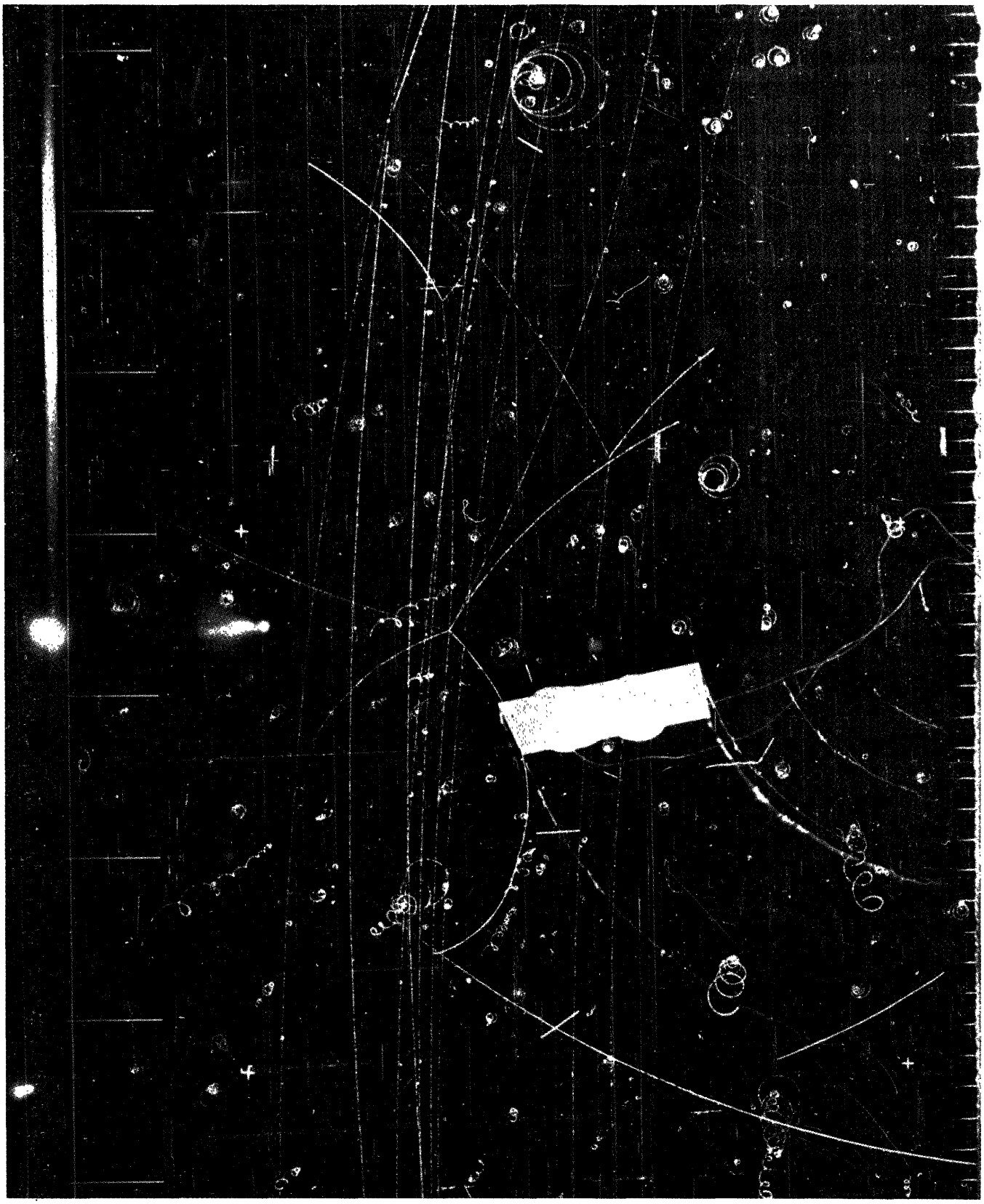


Fig. 2. An enlargement of the Single Vertex Event appearing in Fig. 1.

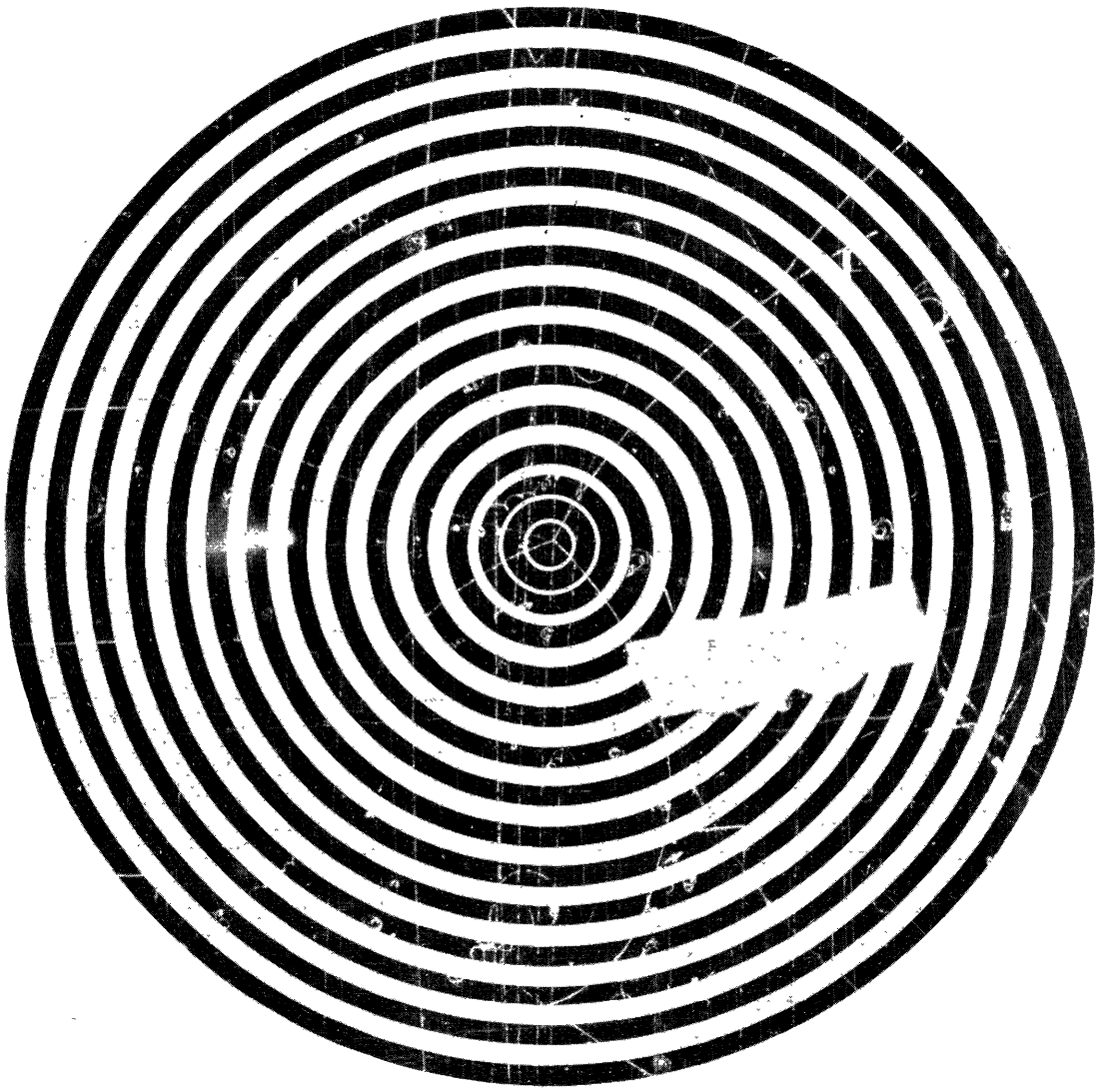


Fig. 3. Annuli seen by Spiral Reader scanning disc.

BLOCK DIAGRAM OF FILTER PROGRAM

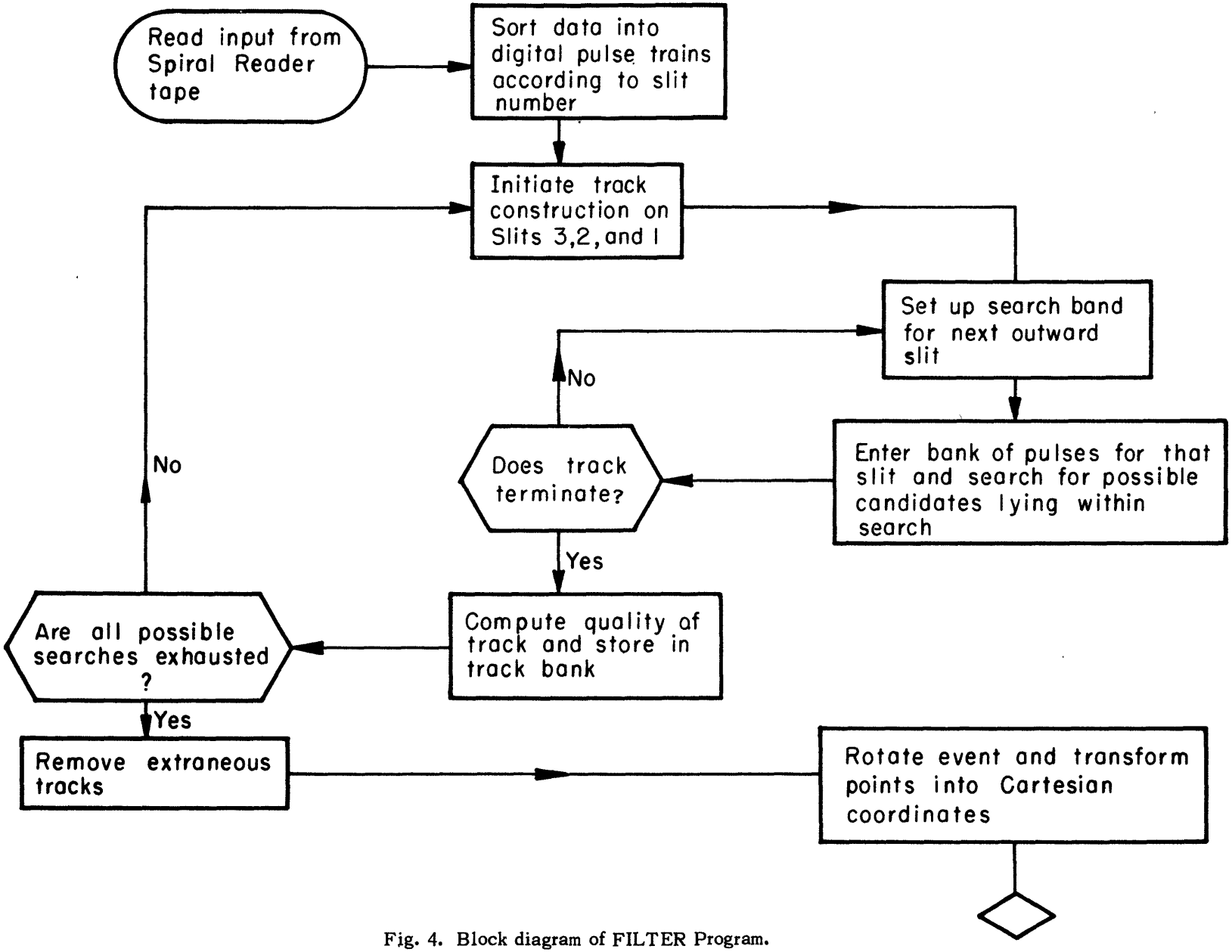


Fig. 4. Block diagram of FILTER Program.

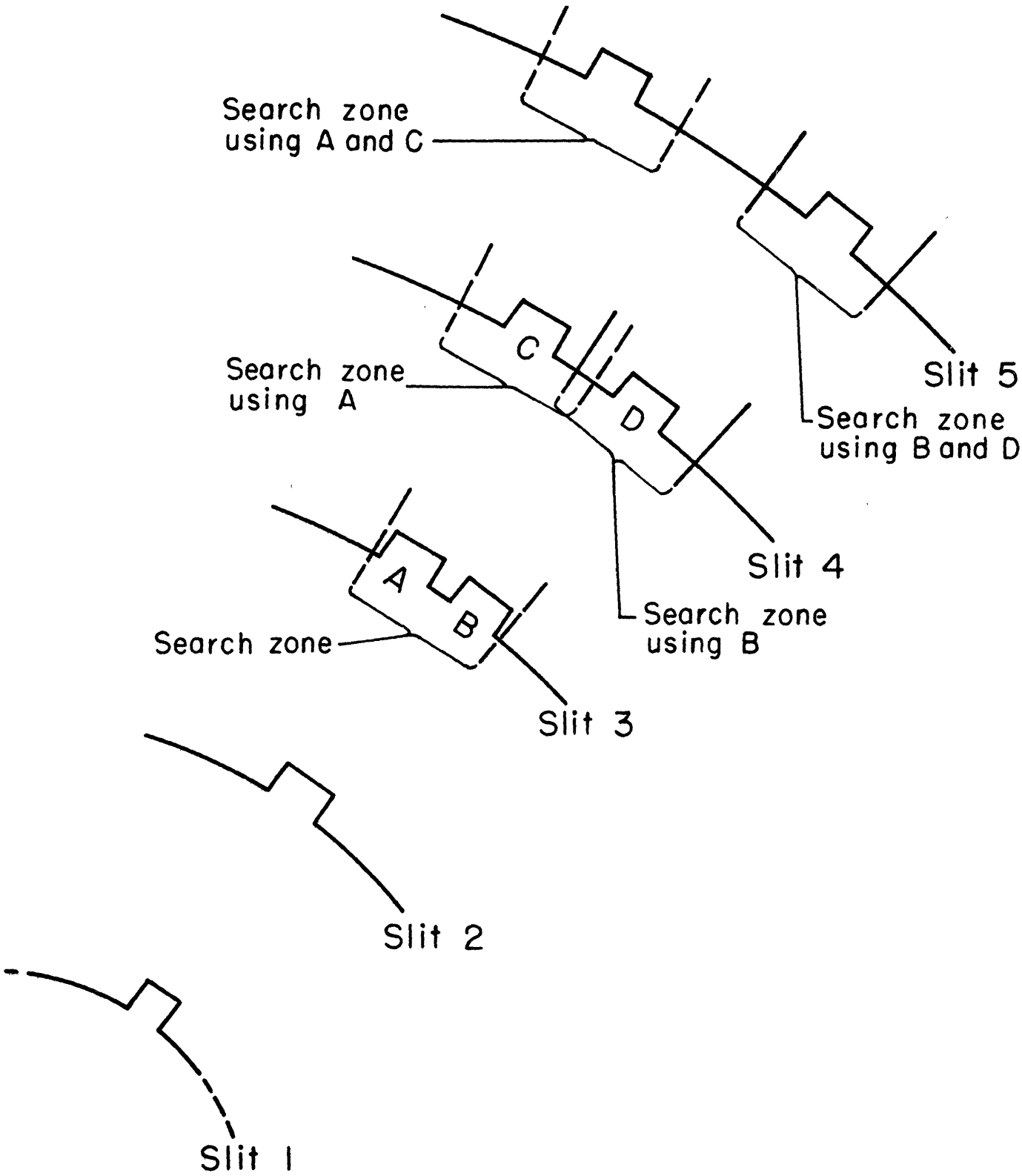


Fig. 5. Branching track pulse trains from first 5 slits.

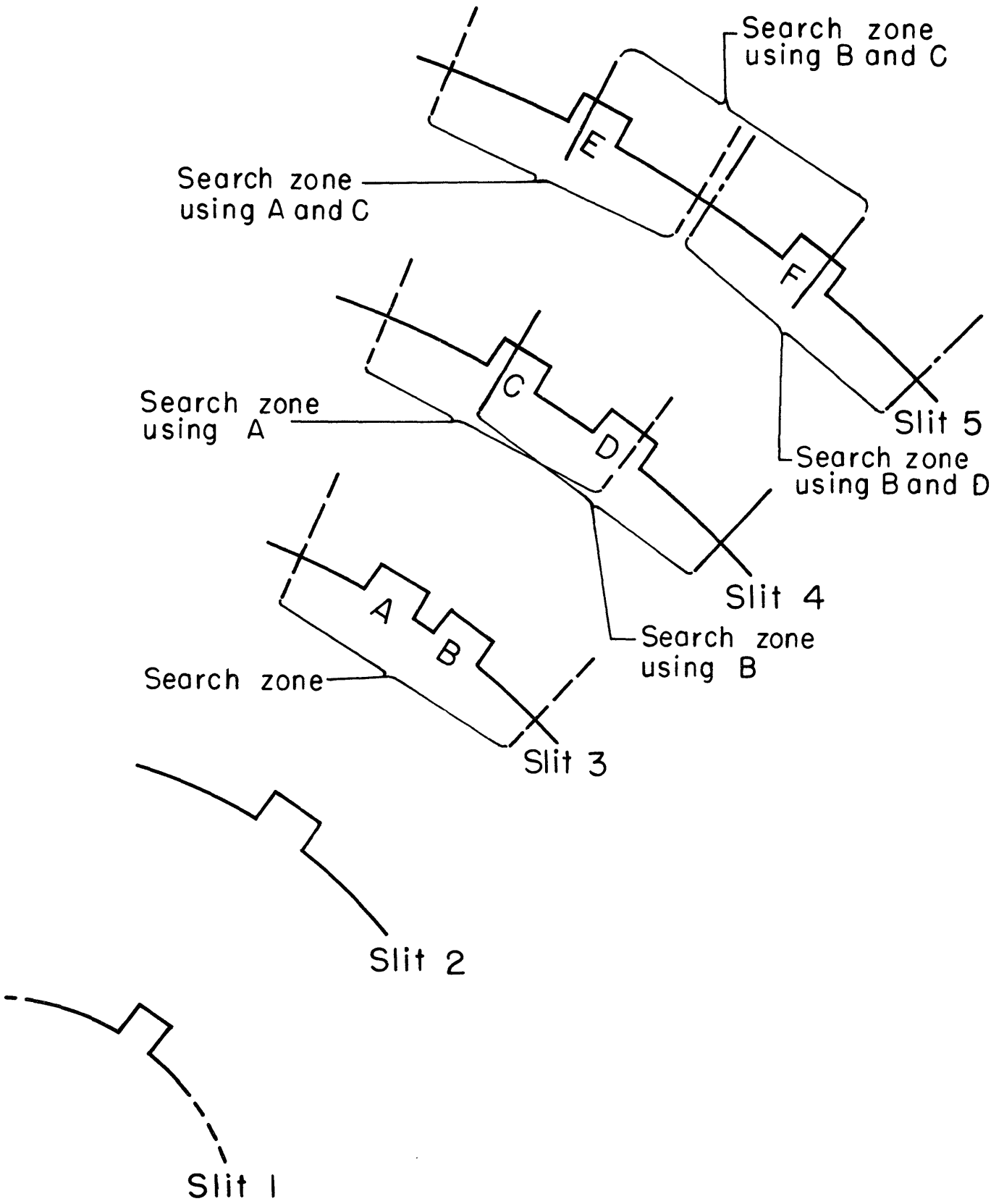


Fig. 6. Effect of doubling search zone width.

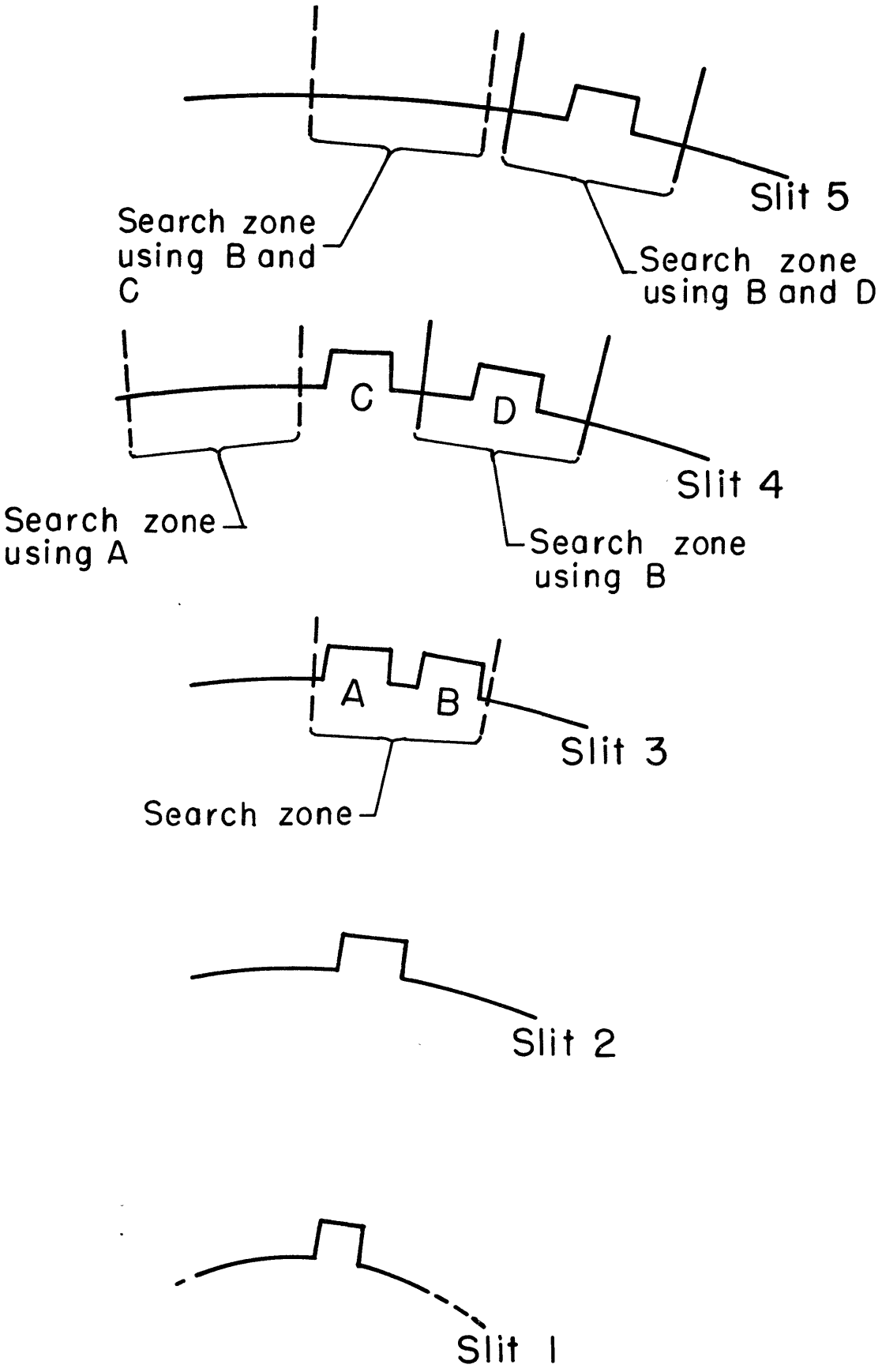


Fig. 7. Effect of close dirt pulse on tracking procedure.

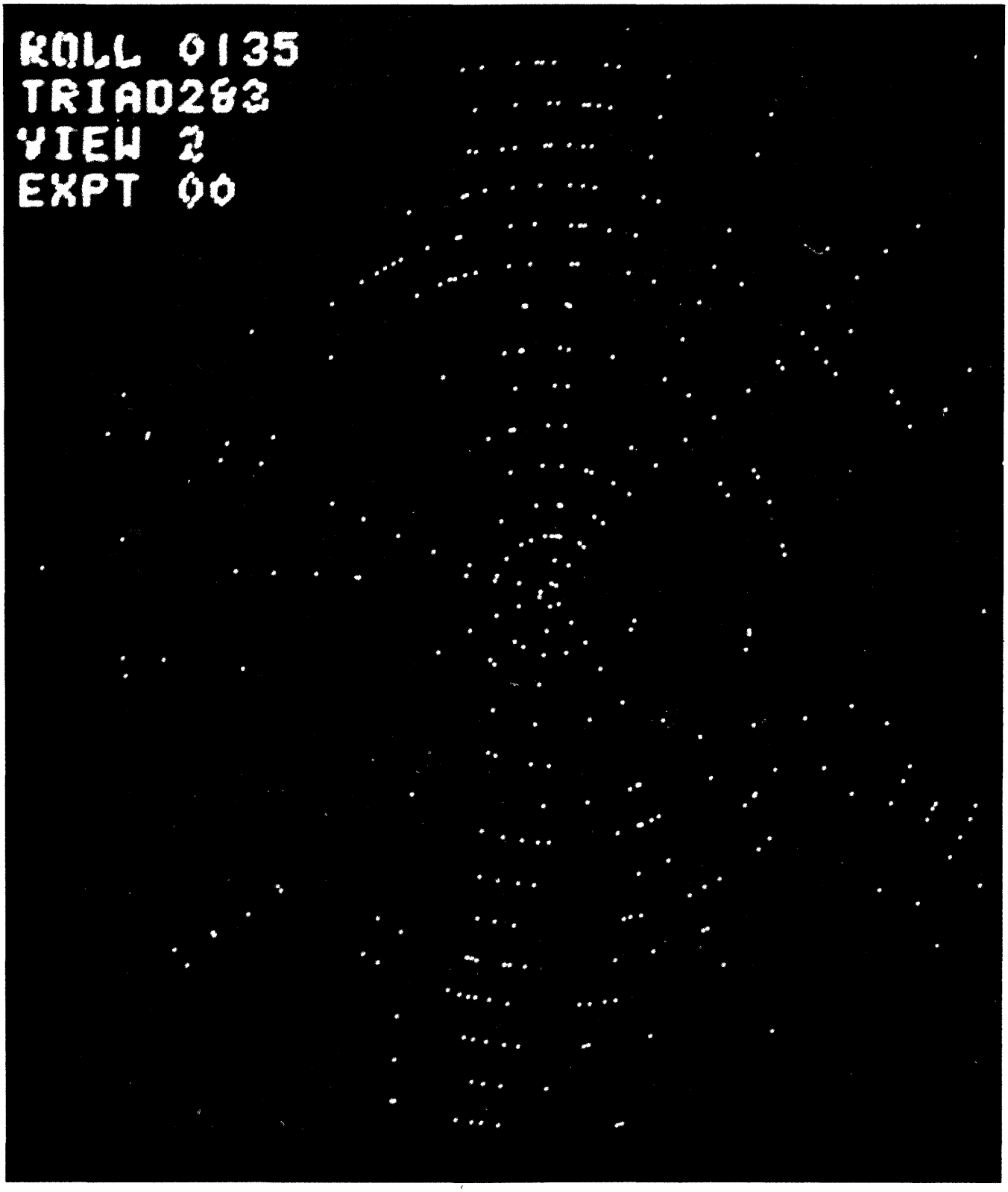


Fig. 8. Cathode-ray-tube display of unfiltered data from Fig. 2 event.

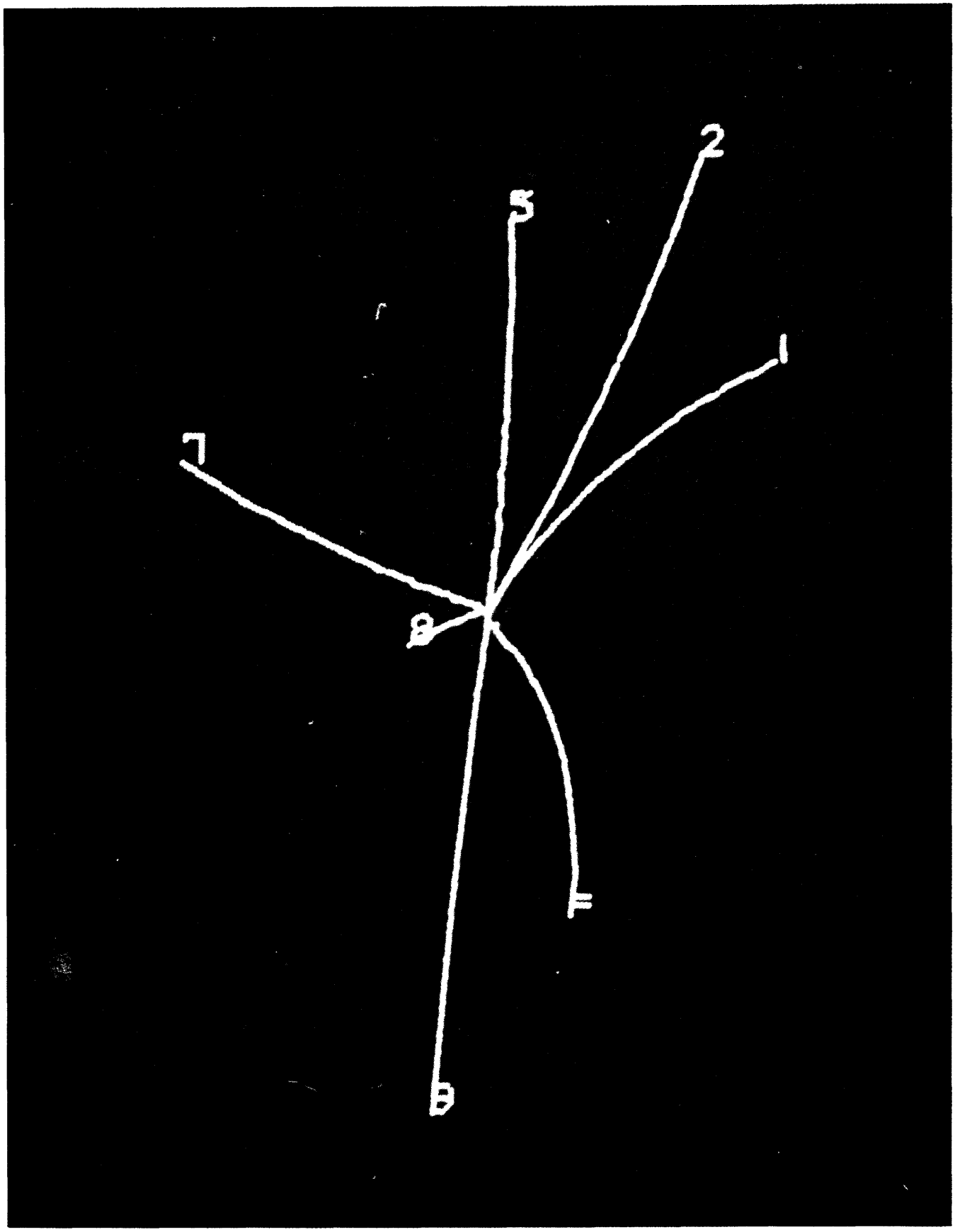


Fig. 9. Cathode-ray-tube display of reconstructed event of Fig. 2.

REDUNDANCY EXPLOITATION IN THE COMPUTER SOLUTION OF DOUBLE-CROSTICS

Edwin S. Spiegelthal, Consultant
Flushing, New York

Double-Crostics, as I suspect many of you know, constitute a species of habit-forming puzzles. There are two varieties of these brain-teasers -- those that appear weekly in the Saturday Review and every two weeks in the Magazine Section of the New York Sunday Times, and a somewhat watered-down version which I have been pitting against an I.B.M. 704 computer. I shall describe both the wild and the domesticated variants in just a moment. First, though, I should like to exercise the air of frivolity which might seem to inhere in my topic. There is, of course, a long and respectable history of fruitful alliances between idle pastimes, on the one hand, and mathematical achievements, on the other. To mention just a few, there are Euler and the Konigsberg bridges, Pascal and his gambler friends, Von Neumann and his poker and, today, Dr. Samuel and his checkers. Still, such is the force of the suspicion that whatever is fun just can't be work, each tentative of this nature must be justified anew on its own merits.

There are at least two good reasons for using a game or a puzzle as an intermediate problem area before one tackles the more mundane problem which one really desires to solve. First, the puzzle or game exhibits the salient features of the honest-to-goodness problem without being encumbered by the inconsistencies and extraneous factors which invariably clutter up the real problem. Second, the puzzle or game has clear-cut rules and, in particular, an unequivocal indication that the puzzle has been solved, or that the game has been won. One question which I shall eventually have to answer in connection with other work I'm doing is: has the machine produced a good translation of this foreign-language text? The mildest thing that can be said about such a question in general is that it does not admit of an unequivocal answer. I do think that the results of my efforts to solve Double-Crostics will aid in producing good machine translations. I don't know if I could have obtained any results had I (and the computer) not been able to say, "this puzzle has indeed been solved."

A third reason for using a well-known game or puzzle rather than a completely neutral mathematical model is that familiarity, contempt-breeder though it may be, does provide the investigator with a corpus of tested problem-solving techniques. These techniques may exist on an intuitive level of cognition only, they may be crudely formulated, they may even be highly inefficient. They do, however, provide a welcome point of departure.

So much for general considerations. It remains to justify the specific choice of Double-Crostics. The relevant factor here is that this type of puzzle does exhibit the salient features of the problem to which I have been addressing myself. This, briefly stated, is the problem of exploiting, by non-algorithmic means, the redundancy inherent in structured-data problems, particularly those problems in which the source data is some natural language or another. Each individual solves this problem for himself more or less instinctively. Computers, unfortunately, do not appear to have this instinct. In certain applications, to be sure, no such instinct is necessary. In a payroll program, for example, the machine knows in advance the kind of information it will receive and the explicit form that this information will take. In machine translation, on the other hand, the machine must extract information of unknown character from running text in which this information is, in general, contained only implicitly. Moreover, to borrow an analogy from electrical engineering, this information usually comes, not in the form of lumped parameters, but as a distributed quantity. The elementary information-bearing particles -- words, in this example -- are generally ambiguous when considered individually, and this ambiguity can be removed, if at all, only by considering the particles "in context." Here again, we all "know" what context exploitation means. In fact, we know it so well, and learned it so young, that we are almost incapable of dredging it up to the conscious level where it must be analyzed before being restated in machine terms, i.e., a program. Let us see now what Double-Crostics are, and how they may help in this dredging operation.

The first slide (Figure 1) is a pretty poor excuse for a Double-Crostic, but it will serve as an illustration. As in crossword puzzles, there is a list of definitions which are supposed to provide adequate clues to the words which are to be filled in. Unlike crossword puzzles, these words are not filled in directly in the nice geometrical pattern off to the right. Instead, they are first filled in next to the verbal definitions and then, character by character, are filled in the so-called Text portion in accordance with the ordinal numbers underneath each character in the Definition portion. The Text portion is so called because, when properly and completely filled in, it spells out a portion of normal English text. The next slide (Figure 2) shows the final state of the Double-Crostic just displayed. The text portion is a fragment from Whittier's poem "Barbara Freitchie". In the

standard variety of Double-Crostic, the first letters of the Definitions, when read from top to bottom, would spell out the author's name and the work quoted, rather than the highly improbable "M. Grisly" which you see.

Let me introduce just a little nomenclature at this point. A verbal definition, "in style" for example, is called a definiens; any of the six-letter words matched with that definiens is called a definiendum, and the plural of definiendum is definienda. We're going to need that plural because, in general, there are several definienda, perhaps many, which a priori match up with a given definiens. Sometimes, however, the puzzle-solver can think of no definiendum for a given definiens. To solve a Double-Crostic, one is always constrained to guess, and to eschew pencils without erasers. The same might be said of crossword puzzles, but the opportunities for sophisticated guessing in Double-Crostics far exceed those in crossword puzzles, where the only correlation between the words filled in is an occasional common character at a point of intersection. In Double-Crostics, let us remember, the Text portion is indeed in English, with all the syntactical and semantic correlations of a natural language fragment. This means that guessing can be applied, not only to the definienda, but to the Text words as well. For example, if the fourth word of the example on the slide had had three of its letters filled in, giving the pattern M _ S T, and the preceding word Y O U had already been filled in, the puzzle-solver would probably feel compelled to guess M U S T, rather than the other contender, M O S T. Frequently, the compulsions are not as strong as this. Still, the fact that a meaningful fragment is to be unearthed, rather than an inchoate collection of unrelated words, does impose a valuable bias on the guesses made. If we turn back to our guess of the letter U to fill out the word M U S T, we note that this guess is not a dead end, but rather the beginning of a chain of decisions. Being a verb form, the word M U S T, if correct, imposes a whole host of constraints, syntactic and semantic, on the remainder of the text fragment. In addition, the letter U is filled in twice, once in the word M U S T, once in the associated definiendum, since it is the third character in the definiendum associated with definiens G, "happy time." This single letter may allow the puzzle-solver to choose a unique definiendum from among the possible definienda. If not, it will at least, in general, shorten the list of possible contenders.

There is another side to the coin of sive guessability, if you will permit the neologism. Due to the inherent redundancy of the data, bad guesses, or bad choices which were not felt to be guesses, will manifest themselves sooner or later. Either a concatenation of implausible or inadmissible characters will appear in a Text word, or a word which is valid in itself turns out to be syntactically or semantically jarring, if not downright wrong. An important point here is that relatively few error indications say categorically that something is definitely incorrect; the typical error is found because it has given rise to a number of sore points, each innocuous in itself but, in their totality, leading one to feel that something is amiss. Another, and operationally more useful, way of stating this is that a correct decision will be maximally consistent with all the other correct decisions to which it relates. Thus, a decision which, while creating no real friction, generates an insufficient quantity of warmth, is thereby suspect.

Although I am speaking about Double-Crostic solution, I believe that much of what I have said applies unaltered to the processes we actually use in understanding natural language texts. I believe that we make the same sort of informed guesses, that we skip back and forth between semantics and syntactics in the same goal-seeking way, that we, consciously or otherwise, apply the same sort of maximal consistency criterion. Thus, I feel that an efficient heuristic technique for solving Double-Crostics will find immediate application to the problems of automating natural text processing, be the process one of translation, indexing or abstracting. Let me then describe the program which does, in fact, sometimes solve Double-Crostics.

Before describing the program, I should say a few words about the kind of Double-Crostic which is presented to the program for solution. It seemed viciously circular to ask the program to "understand" the verbal definiens, since such an epistemological feat would be a pleasant by-product of my endeavors to solve Double-Crostics, but could hardly be a necessary prerequisite for such solution processes. I therefore adopted the alternative of supplying a list of definienda for each definiens of a target Double Crostic as part of the input to the program. Each such list may be vacuous, and the program does not assume that the correct definiendum is necessarily contained on a non-vacuous defienda list. To put it anthropomorphically, the program merely hopes that it will be supplied with enough good information to permit it to achieve the correct solution. Thus, for each Double-Crostic to be solved, the program receives, as input, a description of the physical appearance of the puzzle, i.e., number of text words, number of characters in each word, the correlations of text characters with definienda characters, etc. It also receives the definienda list for each definiens. A few other input parameters are used to limit the time and the storage space used by the solution process.

The program itself consists of two physically, logically, and functionally disparate portions. The first of these remained more-or-less constant during the year or so I devoted to this work. This portion combined the functions of dictionary, grammar book, recording secretary, pencil and eraser, file clerk -- in short, it was the executive organ, the enforcer, of the program. The second portion was the policy-making organ, the brains of the mob, to continue the metaphor. And, like its underworld counterpart, it could easily be replaced if a more likely candidate showed up. Indeed, the present occupant of this spot in the program is the seventh in a series. It is also sufficiently successful for me to consider it as definitive, allowing me to move on to less recondite research tasks.

What resources were available to this policy-making organ? To answer this, we have to take a closer look at the permanent portion of the program. Since it was hoped that the program would be a successful puzzle-solver even though supplied with inadequate and sometimes incorrect information, no attempt was made to provide it with anything like the lexical, grammatical and semantic sophistication of even the dullest human being. Specifically, the text dictionary of the program contains 8551 words, none longer than eight letters. This is supplemented to a certain extent by a subroutine which can decide that a word like "untidily", for example, is probably a valid word, although not in the dictionary, since it consists of a standard prefix "un", a standard suffix "ily" and a dictionary entry "tidy" as a properly modified root. This is the only grammatical "knowledge" in the program's arsenal. Four tables constitute the remainder of the programs familiarity with the English language. The first of these is a list of the hundred most frequently used English words, in the order of their relative frequency. The second is a list of the letters of the alphabet, again in the order of their relative frequency of use. A third table contains the most frequent two-letter combinations into which a letter enters, again ordered by frequency. The fourth table is not an ordered one, but simply consists of a not quite exhaustive listing of the permissible three-letter combinations of English text. What is not explicit in the tabular data, although the program assumes it, is that the dictionary is virtually exhaustive for words not longer than three characters.

In addition to this limited information content, the program possesses the rudimentary pattern recognition facility of perceiving that, as in our earlier example, both M U S T and M O S T provide partial matches with the pattern M _ S T. It might be suspected that, with this restricted intellectual equipment, the program would have only a tiny repertoire of actions open to it. This suspicion is eminently well-founded. Basically, there are only three categories of acts which the program can perform in filling characters in a target Double-Crostic. The next slide (Figure 3) summarizes the situation. There are three essential parameters: whether the word in question is a Text word or a definiendum; whether the word is blank, partially filled or completely filled; whether the word's possibility list is vacuous, contains one entry or contains more than one entry. One possibility list is made up for each word. Initially, the possibility lists for the definiens consist of the definienda lists included in the input, while the lists for the Text words are all vacuous. After a sufficient number of characters have been filled in a Text word, due to actions on the Definition side of the puzzle, its possibility list is filled with all those dictionary entries which give partial matches to the character pattern in the word. As subsequent characters are filled in the same Text word, those possibilities which no longer give partial matches are expunged from the list. Conversely, if any characters are erased from a Text word, its possibility list is augmented by any new partial matches which might thereby be revealed. On the Definition side, the possibility lists can only be decreased, never augmented.

With the foregoing in mind, let us look at the slide. We see that no action can be taken when a word is completely filled, which seems only reasonable, or when the possibility list for the word is vacuous which, I must admit, also seems reasonable. Further, since blank Text words are not permitted to have non-vacuous possibility lists, nothing can be done with them. As for the remaining cases, it does not seem untoward, when there is a single possibility for a given word, two kinds of action are open to the program. It can fill in those characters, if any, which are common to all the possibilities. Thus, for example, there are two five-letter words which begin with W E and which have C for their fourth letter, namely, W E N C H and W E L C H. Both these words have the letter H for their fifth character. Filling in of such common characters is on a par with filling in a full word which is a unique possibility list entry. Sooner or later, however, the supply of such seemingly warranted actions dries up, and the program must begin to guess. The program makes its guesses on the basis of two assumptions: both the Text words and the definienda are good English words, and the Text portion of the puzzle is a reasonable English sentence. The program will thus tend to guess a high-frequency word, on the Text side of the puzzle or, failing that, the possibility on a given list which has, in some sense, the most frequent letters. Specifically, the letters of the alphabet are ranked, with E, the most frequent, being scored 26 and Z, the least frequent, being scored 1. The scores for each possibility are added, character by character, and the highest scorer is guessed.

So ends the catalog of positive actions open to the program. On the negative side, when the eraser must be wielded, two kinds of criteria are first brought to bear. The first kind is close to what we mean when we speak of error detection, since it is applied to manifestations that are patently, or probably, wrong. Using this criterion, the program can decide to erase a definiendum because it gave rise, on the Text side of the puzzle, to too many unacceptable three-letter combinations and/or too many invalid words of three characters or less. As to what constitutes "too many", this is an empirically determined parameter whose variation, during the testing of the program, provided interesting but, unfortunately, inconclusive results. Let me therefore move rapidly to the second species of criterion. Here we have to do with the concept of maximal consistency which I mentioned earlier. This criterion comes into play when the solution process is ready to call a halt, either because all the characters have been filled in, so that the puzzle is apparently solved, or when no further positive actions are open to the program even though some characters remain blank. In both cases, though with perhaps greater urgency in the second case, the program asks itself if each word it had filled in were sufficiently productive, in terms of good two-and three-letter combinations and valid full words on the other side of the puzzle. If any words are found which did not give at least some minimum of consistency with the remainder of the puzzle, the least "fruitful" of these is erased. Here again, the relevant threshold values are parameters at the researcher's disposal.

In the previous section, dealing with negative actions, and when and why they are taken, I moved from the discussion of the permanent program into the realm of the policy-making program, which alone decides what and when. Since the efficiency, if not the efficacy itself, of a heuristic process depends heavily on both the decisions taken and their timing, I have called these policy-making programs "Decision Sequencers." It is the seventh, and last, of these Decision Sequencers which I am discussing now.

After the input describing the next target Double-Crostic has been read into the computer and ingested by the program, Decision Sequencer 7 begins operation. It first takes all the obvious actions, i.e., it scans each definienda list in turn. If there is but one definiendum on a given list, it is filled in. If there are any common characters on a multi-possibility list, they are filled in. After this first pass, the Decision Sequencer enters a decision-making loop around which it continues to iterate until it claims to have found a solution or until it admits defeat. As noted earlier, the entire solution process can be also cut off abruptly if time and/or space run out, but such a halt is, at least logically, independent of the decision-making scheme.

The decision-making loop has such a simple structure that I don't think I'll frighten anyone if I show its flow chart (Figure 4). There are many more lines and boxes I could have included, but I have learned that cute programming is not really a fit subject for public exultation. There is another sense, however, in which I could have included many more lines and boxes without any reference to the underlying programming. I could, that is, have chosen a far more complex logical structure for the Decision-Sequencer. For example, I might have followed the break-through-exploitation philosophy. This would have entailed a tree-like structure wherein each successful, or apparently successful, move would suggest one or more further moves, each of which might in turn suggest a family of moves, etc. Here, the program would have to swoop up and down the tree until all the branches had been exhausted, or until the appearance of a major error would cause the program to retravel some, or all, of the tree in the reverse sense. This may indeed be a better processing scheme. There may be many schemes as good as, or better than, the one I chose. Nonetheless, I had to choose some scheme, I chose a simple one, and it is seemingly a good one. Let me then discuss it.

The simplifying idea underlying the scheme is due to Richard Bellman's book, "Dynamic Programming." The idea, as I have used it, may be stated as follows: the best way to implement a multi-stage decision-making process is, at each stage, to make the decision which is optimal for that stage. In terms of the flow chart, a new stage in the solution process is reached after the previous fill or erase decision has been implemented. At this new stage, only certain positive actions are possible. Each of these possible actions is scored, and the highest-scoring action is fetched so that the attempt to execute it may be made. To clear up confusion, let me hasten to remark that a possible action may not be implementable. For example, if a given definiens has several definienda on its possibility list and no prior attempt to fill in common characters has been made, the present attempt to fill in common characters is a possible action. If there are in fact no common characters, then this possible action cannot be implemented. If an implementable action is eventually found, I consider it to be the optimal action at the present stage.

Once the optimal action is implemented, the new status of the puzzle is verified. What this rather poorly chosen word connotes is a check of each word in the puzzle, wherein poor three-letter combinations and invalid short words are noted disapprovingly, and good two-and three-letter combinations and full words receive due reward, "gold stars" being marked down for them

in a special record. I will have more to say about these gold stars when I discuss the key process of action scoring. After the gold stars have been handed out, the program glances over the puzzle to see if, perchance, all the characters have been filled in. If they have not, the next stage of the decision process is initiated by the re-entry of the program to the uppermost box on the flow chart. The little boxes lying off the path I have just beaten were discussed earlier when I spoke of the negative actions open to the program.

It is almost time to discuss the heart of the Decision-Sequencer, namely, the scoring process and its associated verification process. First, though, I should like to answer an objection which might be raised at about this point. I am presumably claiming that my consistency criterion allows me to rank finished Double-Crostics as to their acceptability. This is certainly implied by my use of a numerical score which I compare with a threshold to determine if a finished puzzle is acceptable. Why then, it might be asked, do I not simply try all the combinations of the given definienda, and accept the best scoring configuration as the solution, thus rendering all this scoring and verification folderol nugatory? In fact, the objector might add, if you would only guarantee that the correct definiendum is included in every definienda list then you could also do away with all your guesswork. Dropping now my role of Devil's Advocate, let me answer the second point by noting that, in the bread and butter type problems which I'm working towards, the analog of the correct definiendum is not, and cannot, always be included in the relevant list. As to the exhaustive approach suggested in the first part of the objection, I have the law of exponential growth on my side. For example, I shall presently show you the solution process for a Double-Crostic somewhat more challenging than the example I first showed. There are only 18 definiens for this puzzle, one of which had three definienda on its list, the other 17 having only two definienda each. This is a total of only 37 definienda, but the number of combinations which I would have to try in accordance with my critic's suggestion is something over 393,000. In the particular solution process I shall sketch later, 33 positive actions were taken, four of which were incorrect and were subsequently cancelled by four negative actions. The total machine time for this solution process, including a detailed tape printout at each stage, was about 30 seconds. There seems then to be some point in employing to folderol which we are about to discuss.

In the scoring process, an essentially qualitative value judgment is given quantitative form. This is by no means equivalent to a transition from subjectivity to objectivity. The best one can do is to minimize the area in which subjectivity operates and to limit the scope of its activities in that area. In the concrete case of Double-Crostic solution, subjectivity impinges in two ways on the scoring process: it affects the choice of the factors deemed to be relevant, and it enters into the assignment of relative weights to those factors. In setting up the earlier Decision-Sequencers, I allowed my intuition free play in both these directions. I "felt" that the relevant factors, in scoring some action for a given word, were the total length of that word, the number of characters already filled in that word and the number of gold stars already accumulated by that word. I then multiplied these numbers by seemingly "reasonable" coefficients, as a function of the particular type of action being scored, and then added a fudging factor, again as a function of the type of action. Oddly enough, I still managed to solve some Double-Crostics this way. However, having no coherent rationale for this scoring scheme, I certainly had no clues as to how it might be generalized so that I might apply it to the problems in which I was really interested. Furthermore, while I could applaud my intuition when the program solved a Double-Crostic, I could not explain the cases where the program failed. I could, of course, blame my intuition, but this would hardly have been constructive criticism. In the seventh, definitive, Decision-Sequencer, the choice of relevant factors is dictated by a coherent underlying philosophy, and the assignment of weighting coefficients, while still arbitrary to a certain extent, is nonetheless constrained by considerations stemming from this same philosophy. I use the term "philosophy" because the concepts are general ones, and must be properly specialized so as to apply to the given concrete case. I cannot pretend to you that the general philosophy has been worked out fully. I shall therefore restrict myself here to its specific application to Double-Crostic Solution.

Viewed cold-bloodedly, a Double-Crostic is seen to consist of two sets of hypothesis sets, each Text word and each definiendum being considered a hypothesis. Initially, some of these hypothesis sets may be vacuous, and the non-vacuous ones need not contain the correct entry. It is assumed, rather vaguely, that an "adequate" number of hypothesis sets do contain the proper entry. An acceptable final state will be one where exactly one hypothesis will have been chosen for each hypothesis set, where some reasonable proportion of the final hypotneses will have occurred as initial hypotheses in their respective hypothesis sets, and where each final hypothesis will be a good English word. The first requirement, of course, simply restates the obvious fact that no puzzle is complete that does not have all its characters filled in. The second requirement imposes the need for one of the threshold values which I have previously discussed. The third requirement can be phrased somewhat more discursively as follows: only

some tiny percentage of Text words less than four characters in length may not match up with dictionary entries; no definiendum may give rise to too few good two-letter or too many poor three-letter combinations on the Text side of the puzzle, while a similar constraint is imposed on the Text words concerning their effect on the Definition side of the puzzle. A human puzzle-solver certainly imposes more constraints on the final solution than those I have enumerated. For instance, he would presumably reject a four-letter combination such as D E X C since no English word contains this particular sequence of letters. The Decision-Sequencer, however, does not check for four-letter combinations, and would find the two triples D E X and E X C to be perfectly legitimate. Similarly, the machine would not take exception to the phrase Y O U I S, although most Double-Crosticians might. The point here is that what is redundancy for the goose can be startling news to the gander. It is in the verification portion of the Decision Sequencer that redundancy factors are checked. It is in the scoring portion that the decisions are made which will tend to give the greatest field of action to the verification process at each step. That is, all other things being equal, the scoring process will, or at any rate should, present the verification process with the most new two-and three-letter combinations, short words and full words as possible. Sheer numbers alone cannot be used. Two pairs of letters, for instance, are not nearly as potent as two triples. On the other hand, two triples are not fifty times as potent as two pairs. Thus, while the relative weights of these two factors can be chosen somewhat arbitrarily, the range of variation is not particularly great.

It should be clear that any scoring process will automatically tend to fulfill the first two requirements on the solution, since the actions which are scored are all positive actions, entailing the filling in of more characters, and are all based on the use of hypotheses which lie in the original hypothesis sets. The burden of meeting the third requirement, the one specifying that the final result will be "good" English, falls on the verification process primarily. Apart from the final glance at the puzzle in which the program reassures itself that the chosen hypotheses have been sufficiently "fruitful", it is the verification section which does the lion's share of "error" detecting. Thus, if efficiency were not an operational requirement, the entire scoring process could be replaced by a random number generator for the selection of the next action to be implemented. Efficiency, however, is a major requirement, perhaps the major requirement. How, then, do we construct an efficient scoring procedure? First, as noted, earlier, we try to present the verification process with that action which, when implemented, will provide the greatest sum of weighted redundancy factors, everything else being equal. Second, we analyze these other things which may or may not be equal. One of these things is the gold star count already compiled by the relevant hypotheses. For example, if several common characters of a given definienda set had already been filled in, leading to all sorts of nice two-and three-letter combinations on the Text side of the puzzle, it would presumably be safer to guess one of the definiendum of this set than to guess a definiendum of another set which had thus far been unproductive. Another cogent factor is the length of the possibility list of each of the contending hypothesis sets. The contention here is that, all other things again being equal, the shorter possibility list should be attacked first. The reasoning here, as you will see, is somewhat tenuous. Given two hypothesis sets, one with two entries on its possibility list, the other with three entries, we might argue as follows. We do not know whether either list contains the correct entry. Therefore, we might just as well assume, for the sake of argument, that the correct entry is contained on each list. If we are fortunate enough to choose the correct entry from the list we select, then it will have made no difference which list we chose. If our choice of entry is not correct, however, we can hope that the verification process will cause its erasure sooner or later. If we erase one possibility from the two-entry list, we are presumably left with the unique correct entry. If, on the other hand, we erase one possibility from the three-entry list, we still have to make a choice between the remaining entries. To this argument we must add the consideration that each positive action on one side of the puzzle will entail a corresponding reduction in the possibilities open to the hypothesis sets on the other side, since the numbers of partial matches will, in general, be decreased. Thus, the sooner a correct choice is made on the Definition side, the sooner will correct choices be possible on the Text side, leading to more correct choices on the Definition side, etc.

One final factor must be taken into account in the scoring process. It is natural, and probably correct, to assume that some kinds of actions are inherently preferable to others. Specifically, choosing a unique possibility, or filling in common characters, would be preferred to guessing a definiendum, no matter how many high-frequency characters it contained. Since the correct definiendum is not necessarily included in the original definienda list and since, on the other side of the fence, the Text dictionary is not complete, the first category of actions can certainly not be considered as sure things. This same uncertainty, of course, attaches to the hypothesis set from which any guessed entry is selected. It still seems reasonable, therefore, to give higher score to the relatively more warranted actions.

The scoring process for a given action and a given hypothesis set can now be summarized. The puzzle is scanned to see what new redundancy factors would be engendered by the action, and each such factor is properly weighted. The sum of all these weighted redundancy factors is then suitably modified by the number of gold stars already accumulated by the given hypothesis set, the length of the set's possibility list, and a warranty factor, which is a function of the type of action being scored. In the case of the action to seek common characters one slight additional modification is necessary. If an action to guess a word of n unfilled characters is made, then all n of these characters will be filled in by the action. On the other hand, an action to seek common characters in a word with n unfilled characters will lead to at most n-1 characters being filled in, and may result in no characters at all being added. It is therefore necessary to multiply the raw score for a common character action by the expected proportion of characters that will result. This proportion can be determined empirically if intuition fails.

If I were to apply the philosophy underlying Decision Sequencer 7 to some other problem area, I would proceed about as follows. I would first determine the hypothesis sets relevant to the problem, together with the available means for changing these hypothesis sets during the solution process. I would next decide on the actions that could be taken in selecting one hypothesis from a given set, and on the criteria for determining the a priori impossibility of a given action at a given stage of the solution process. Third, I would determine what redundancy factors relating the hypotheses I would or could use, and what relative importance I should attach to each such factor. Finally, I would have to decide on the stop rules, i.e., I would have to specify when the solution process was to be considered either successfully completed or impossible to complete. I cannot, at this date, report on whether this prescription has cured, or killed, any patients, I hope to report the former at some later date.

As a sort of appendix, I should like to present a few selected moments from three actual solution processes. The first could not be completed by the program. The second and third could, which is the reason why I conclude the paper with them. The next slide (Figure 5) shows the puzzle status at a point when any red-blooded American could finish it with his left hand tied behind his back. Prior to this point, the program had made four different bad guesses which it was subsequently able to catch and erase due to the exorbitant number of poor three-letter combinations that had ensued. At the point which now concerns us, the program suffered from the paucity of lexical information with which I provided it. The program was aware of only one six-letter word which matched with T R __ H _ and was just unaware that the sixth letter could have been, let alone should have been, the letter S. It therefore completed this word with its unique possibility, T R O P H Y. The program was unable to rectify this particular error so that, after some thrashing about in which some earlier correct choices were despairingly erased, the program ground to a halt with the solution carried to the point shown on the next slide (Figure 6). The Text does have a certain piquancy, but it is undeniably incorrect. Two points are to be noted in connection with this failure. This first is that it could have been avoided had I taken slightly greater pains in providing the program with its basic lexical and grammatical lore. The second point is that the solution process, before being stalled, had proceeded extremely rapidly. The definienda lists provided for this solution process contained over 145,000 combinations. The correct status shown on the previous slide had been arrived at after seventeen positive decisions had been made, including the four erroneous ones which had been corrected prior to the point in question. I am therefore not too discouraged by this particular failure.

Still, I find success particularly encouraging. The next slide (Figure 7) shows the status of the target Double-Crostic just before erasure of the fourth, and final, error made by the program. As in the previous case, the dictionary was inadequate, allowing the incorrect unique partial match C O N S U M E R to be filled in, rather than the correct word C O N S I D E R which was not included in the dictionary. Fortunately, C O N S U M E R gave enough poor three-letter combinations on the Definition side of the puzzle to warrant its erasure. Since the erasure of any character automatically leads to the erasure of all characters to whose filling in the erroneous character made some contribution, the erasure of C O N S U M E R led to the significantly stripped-down status shown in the next slide (Figure 8). It was clear sailing from this point on, however, so that fifteen decisions later the final, and correct, puzzle status was reached, as shown on the next slide (Figure 9). Although the program remains oblivious to the extra redundancy involved, the first letters of the correct definienda here do spell out the name of the author of the Text fragment and the title of the source, to wit, the Memoirs of Harry Truman (Volume I, page 189 to be exact). If I may be forgiven for repeating myself, this puzzle was solved by means of 29 correct decisions and 4 incorrect ones which were subsequently rectified.

In this particular solution process, the verification section was particularly stern, in that a definiendum would be erased if it engendered only two poor three-letter combina-

tions or just one inadmissible short text word. Out of curiosity to see what would happen if I went to the other extreme, I disabled this portion of the verification section completely, thus allowing all triples and short words to be considered admissible. This left the program, as its sole error-detecting capability, its facility for rejecting final configurations which were not minimally consistent. The next slide (Figure 10) shows the final configurations which were presented for inspection when the previous Double-Crostic was used as a target. The fifth time around the correct configuration was presented and was accepted. As in the previous case, four incorrect decisions had been made here, but only 20 correct decisions had to be made. I would not like to generalize on the relative merits of the two forms of error-detection on the basis of such a minuscule sample. I think it fair though to conclude, on the basis of all the testing I have performed, that the combination of redundancy factor checking and consistency inspection, taken in conjunction with scoring by weighted sums of redundancy factors, as modified by list length, gold star count, warranty factor and expectancy factor, will provide effective decision sequencers in more urgent, if less entertaining, problem areas than Double-Crostics. Time will tell.

NOTE: Permission to refer to Double-Crostics granted by Saturday Review; permission to quote from the Memoirs of Harry Truman granted by Time, Incorporated

FIGURE 1 -- Double-Crostic 1 (Original Status)

Definitions

A. In style — 11 9 21 6 1 26

B. Common ailment — 22 4 10 15

C. Concerning — 23 27

D. Personal — 17 29

E. Weakling — 18 3 7 14 8

F. Asian spot — 20 2 28 13 24

G. Happy time — 25 19 12 5 16

Text

1A 2F 3E 4B 5G 6A 7E 8E 9A 10B 11A 12G 13F 14E

15B 16G 17D 18E 19G 20F 21A 22B 23C 24F 25G

26A 27C 28F 29D

FIGURE 2 -- Double-Crostic 1 (Final Status)

Definitions

A. In style — M/11 O/9 D/21 I/6 S/1 H/26

B. Common ailment — G/22 O/4 U/10 T/15

C. Concerning — R/23 E/27

D. Personal — I/17 D/29

E. Weakling — S/18 O/3 F/7 T/14 Y/8

F. Asian spot — L/20 H/2 A/28 S/13 A/24

G. Happy time — Y/25 O/19 U/12 T/5 H/16

Text

S/1A H/2F O/3E O/4B T/5G I/6A F/7E Y/8E O/9A U/10B M/11A U/12G S/13F T/14E

T/15B H/16G I/17D S/18E O/19G L/20F D/21A G/22B R/23C A/24F Y/25G

H/26A E/27C A/28F D/29D

FIGURE 3 -- Possible Positive Actions as a Function of Word Parameters

Status of Word	Blank		Partially Filled		Completely Filled	
Number of Entries on Possibility List	Text	Def.	Text	Def.	Text	Def.
Zero	X	X	X	X	X	X
One	X	Fill Unique Entry	Fill Unique Entry	Fill Unique Entry	X	X
More Than One	X	Seek Common Char-acters; Guess	Seek Common Char-acters; Guess (h-f Word or h-f char-acters).	Seek Common Char-acters; Guess	X	X

FIGURE 4 -- Decision Sequencer 7, Simplified Flow Chart

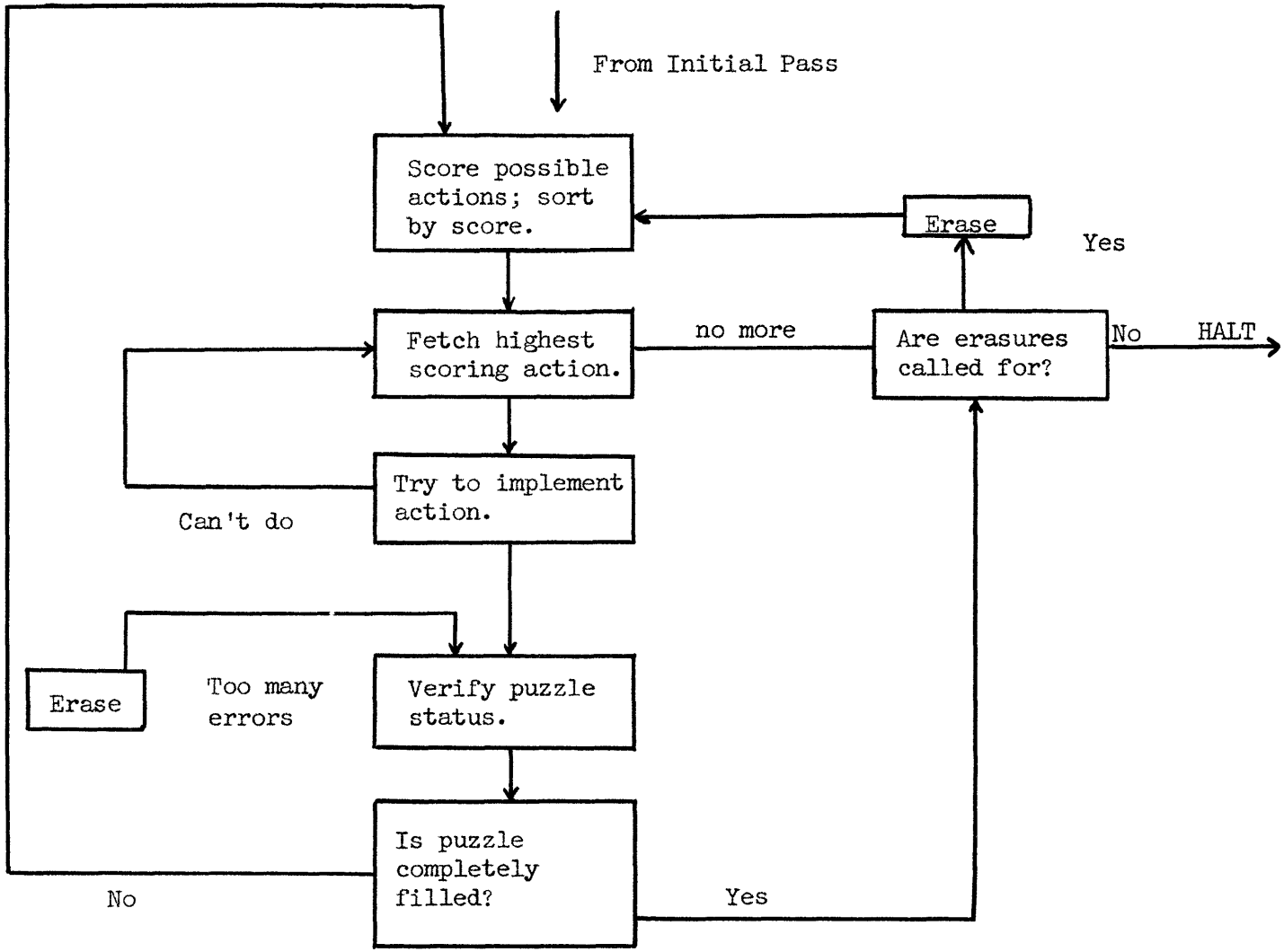


FIGURE 5 -- Double Crostic 2, Intermediate Status

Definitions

A. _ _ _ N _ _

B. C O W B E L L

C. H A L V E S

D. F O D D E R

E. T H I R D

F. T E E T H E

G. H E L M E T

H. T R U A N T

I. _ A _ E

J. S _ _ _ _

Text

W E H O L D _ H E S _ T R _ _ H _ T O B E S E L F

E V I D E N T T H A T _ L _ M E N A R E C R _ _ T E D

_ _ U A L.

FIGURE 6 -- Double Crostic 2, Final Status

Definitions

A. S O O N T _

B. C O W B E L L

C. H A L V E S

D. F O D D E R

E. T H I R D

F. T E E T H E

G. H E L M E T

H. T R U A N T

I. E A Y E

J. S D _ U P

Text

W E H O L D T H E S E T R O P H Y T O B E S E L F

E V I D E N T T H A T O L D M E N A R E C R _ _ T E D

U S U A L

FIGURE 7 -- Double Crostic 3, First intermediate Status

Definitions

A. H E G E M O N Y

B. A B A C U S

C. R E T C H

D. R A B I D

E. _ _ _ _ _

F. T O N I C

G. R _ S I N

H. U T M O S T

I. M N E M O N I C

J. A N A T H E M A

K. _ _ E _

L. _ _ M D I _ _

M. E T H I C S

N. M O _ _ _

O. _ _ _ _ _

P. I N I T I A T E

Q. R E V I S I T

R. _ U _ E

Text

I C O N S U M E R T H E M E T H O D S U S _ _ B _ T H E

_ _ _ S E _ O M _ I _ T E _ O N _ N _ M E R I C A N

A C _ I V I T I E _ _ O B _ T _ E M _ _ T U N A M _ R I C A _

T H _ N G I _ A M E R I C A I N I T S D A Y.

FIGURE 8 -- Double Crostic 3, Second Intermediate Status

Definitions

A. H E G E M O N Y

B. A B A C U S

C. R E T C H

D. _ _ _ _ _

E. _ _ _ _ _

F. _ _ _ _ _

G. R _ S I N

H. _ _ M _ _ _

I. M N E M O N I C

J. A N A T H E M A

K. _ _ _ _

L. _ _ _ D _ _ _

M. _ _ _ _ _ S

N. _ O _ _ _

O. _ _ _ _ _

P. I N _ T _ _ _ _

Q. R E V I S I T

R. _ _ _ E

Text

I _ _ N S _ _ E _ T H E M _ _ H O D S _ S _ _

_ _ T H E _ _ _ _ E _ O M _ _ _ _ E _ _ N

_ _ _ M E R _ C A N A C _ _ V _ _ I _ _ _ O B _ T _ E

M _ _ T U N A M _ R I C _ _ _ _ _ N G I _

_ _ _ R _ _ A I N _ _ S _ A Y.

FIGURE 9 -- Double Crostic 3, Final Status

Definitions

A. HEGEMONY	J. ANATHEMA
B. ABACUS	K. NEED
C. RETCH	L. MIDDIES
D. RABID	M. ETHICS
E. YOUTH	N. MOUTH
F. TONIC	O. OCEAN
G. RESIN	P. INITIATE
H. UTMOST	Q. REVISIT
I. MNEMONIC	R. SITE

Text

I CONSIDER THE METHODS USED BY THE HOUSE COMMITTEE ON UNAMERICAN ACTIVITIES TO BE THE MOST UNAMERICAN THING IN AMERICA IN ITS DAY.

(Memoirs of Harry S. Truman, Volume I, page 189.)

FIGURE 10 -- Double Crostic 3. Four Final Configurations Successively Rejected by the Consistency - Checking Section

1. I CONSIDIS TDE METHIDS USED EY THE HOISE CONMRITTEE
   ON UNAMORISAN ACTIVITIES CO RE TEE MOST EOAMERICAN
   THINM IN TVERICA IN ITR NEN.

2. I CONSIDES THE METHODS USED EY THE HOISE COMMRITTEE
   ON UNAMERISAN ACTIVITIES CO RE TEE MOST ENAMERICAN
   THING IN TVERICA IN ITR NEY.

3. I CONSIDES THE METHODS USED EY THE HOISE COMMRITTEE
   ON UNAMERICAN ACTIVITIES CO BE TEE MOST UNAMERICAN
   THING IN TVERICA IN ITS NAY.

4. I CONSIDES THE METHODS USED EY THE HOUSE COMMRITTEE
   ON UNAMERICAN ACTIVITIES TO BE THE MOST UNAMERICAN
   THING IN TMERICA IN ITS NAY.

A COMPUTER FOR WEATHER DATA ACQUISITION

Paul Meissner, National Bureau of Standards, Washington, D. C.
James A. Cunningham, National Bureau of Standards, Washington, D. C.
Claude A. Kettering, U. S. Weather Bureau, Washington, D. C.

Summary

A need for improved reporting of weather data has been brought about by the requirements of modern, high-performance aircraft, together with the advent of high-speed computers for use in weather forecasting. Manual methods of recording meteorological observations introduce an undesirable time delay, increase the chance of error, and limit the frequency of observations. A solution to this problem lies in the use of automatic data processing equipment for the recording, pre-processing, and transmission of the information. Under the sponsorship of the U. S. Weather Bureau, the National Bureau of Standards has developed a specialized computer for use as a research tool in exploring this concept.

Introduction

The National Bureau of Standards in cooperation with the U. S. Weather Bureau has developed a specialized digital computer for use by the Weather Bureau as a prototype in the development of automatic weather stations. This computer receives data from weather-sensing instruments and processes these data through such functions as sampling, comparing, selecting a maximum, and arithmetic operations. The results operate local and remote displays, and are transmitted via teletypewriter to a central forecasting station and to other airport weather stations. Values of two quantities recently developed as aids to air safety--runway visual range and approach light contact height--are given by the machine through automatic table look-up.

Increased use of weather reports by the general public and the aviation industry, and others, has placed a demand on the Government to provide more frequent and more accurate weather data from active airports and remote locations. It has been obvious that this demand could best be met by the development of automatic meteorological equipment rather than by increasing the number of observers. One can readily appreciate the advantages of having automatic weather stations capable of operating unattended for long periods. Such stations could be established in regions not normally habitable, but which might nevertheless be important from a meteorological standpoint. Even at manned stations, the use of automatic equipment offers many advantages. Station personnel are freed from routine observations, yet readings can be taken more frequently, with less chance for error, and transmitted with less delay. A single eight-hour shift might suffice for the personnel, at locations now requiring continuous attendance. The United States Weather Bureau has, therefore, supported the development of automatic weather stations since the end of World War II.

The first of these stations to be developed and operated in service was used in the British West Indies and transmitted a limited amount of meteorological information to Miami, Florida, by radio, during the hurricane season. As telemetering techniques were developed, it was possible to increase the amount of data transmitted. Further development has provided stations now in use, which transmit weather observations on demand into the National Weather Network.

These stations demonstrated the practicability of automation in collecting meteorological data, but provided no computing capability and insufficient versatility to meet changing requirements of the meteorologist. In order to provide a complete report automatically, certain computing and decision-making capabilities are required. These requirements, and those from other Government Agencies, have led to a joint project between the United States Weather Bureau and the Data Processing Systems Division of the National Bureau of Standards, in which meteorologists and engineers have worked together to develop an Automatic Meteorological Observing System with both operational and research capabilities. The present equipment, designated AMOS IV, is built around a small, specially designed general-purpose computer, to which have been added the required input-output facilities.

Requirements of the Automatic Station

Consider the tasks which must be performed by an automatic weather station. The station is equipped with a number of weather-sensing instruments which furnish weather data, in a variety of forms. Data from the instruments must be suitably processed to obtain the desired information, and this must be made available in the correct form for display and transmission. It is necessary to assemble the information, together with any additional material, such as the station code and remarks, in the correct format for several different output messages. These messages, are to be available for the teletypewriter transmission upon receipt of command signals.

In a few cases, instrument readings could be sampled directly, converted to teletypewriter code, and transmitted. However, in many cases, the desired quantities are not suitably represented by the instantaneous instrument readings. Hence, varying amounts of processing are required. In previous AMOS prototypes, the required intermediate processing has been achieved through the use of a variety of separate devices. In some cases, requirements have developed to the point where computer-type equipment is required, although individually the various instruments do not fully utilize the circuit capabilities.

It became apparent that the overall hardware could be reduced through the use of a central data processor which could be time-shared by the various instruments. The use of an internally-programmed machine for this purpose greatly increases the versatility of the automatic station, permitting more sophisticated processing of data from all instruments. The meteorologist is afforded the opportunity to arrive at optimum data-processing routines, comparing different procedures simultaneously, if desired. Changing requirements may be met simply by preparing new programs; thus, there is an inherent guard against obsolescence.

For illustrative purposes, a number of quantities of interest and the associated instruments will be described, together with the form of output obtained and the processing required.

Transmissivity

Transmissivity of the atmosphere is measured by a transmissometer. A horizontal beam of light of known intensity is directed at a detector several hundred feet away. The amount of light received controls the pulse rate of an oscillator. The pulse rate varies from nearly zero, for heavy obscuration, to about 4000 pulses per minute with a very clear atmosphere. A small background count may be obtained with no light at all; by periodically turning off the source this background count may be obtained and subtracted as a correction factor. The transmissivity data is used in two forms. For some uses it is expressed as a percentage of the maximum obtainable value. Thus, a pulse rate of 3000 ppm would indicate a transmissivity of 3/4, or 75%. On the other hand, the corresponding visibility in miles is not a linear function and is different in the daytime than at night. A pulse rate of 3000, for example, represents a visibility of 1.5 miles in the daytime and 2.4 miles at night.

Wind Speed

Wind speed data is received from an anemometer in the form of a pulse rate. The anemometer produces 5 pulses per second for each knot of wind speed. There are three quantities which we wish to derive from the pulse rate; these are:

(1) the peak one-second gust occurring over the past ten minutes,
(2) the one-minute average wind speed,
(3) the ten-minute average wind speed,

These quantities are to be updated each minute.

Temperature

Temperature is measured by a bridge circuit which is kept in balance by means of a servo-operated slide wire. The servomechanism also operates contacts, to furnish a read-out of three decimal digits and sign. Each digit is represented by a contact closure on one of ten wires. Temperature data is transmitted as read, and in addition is used as a correction factor for certain other data.

Pressure

Pressure is sensed by a mercurial barometer which has a small magnetic float riding on the top of the mercury column. A servomechanism maintains the position of a sensing coil with respect to the float, and operates contacts similar to those of the thermometer. A read-out of four decimal digits is obtained. Several quantities are of interest in addition to the current pressure. For aviation use an altimeter setting must be obtained. This can be obtained electrically, by means of additional contacts, but can also be handled by the computer via table look-up or calculation. The pressure must be converted to an equivalent sea-level value, and this requires a temperature correction using two temperature values spaced 12 hours apart. Pressure tendency is another calculation and consists of examining three values taken at hourly intervals. From these a coded value for the trend is obtained.

Cloud Height

Cloud height data is obtained from a ceilometer. A beam of light from a rotating searchlight is projected on the clouds and the amount reflected is measured by a photocell. Cloud height is obtained by triangulation, using the distance between the searchlight and the detector, together with the angle of the searchlight when a cloud signal is received. In order to obtain cloud height automatically, it is necessary to use two circuits, one which watches for peaks in the photocell signal, while the other keeps track of the searchlight angle. Whenever a cloud signal is indicated by the photocell, the corresponding angle is stored in a buffer register. Height can be obtained from the angle by table look-up or calculation. It is desirable to store a number of cloud observations and scan the stored data to answer such questions as the following:

(1) At what height was the predominant cloud activity observed over the past ten minutes? (This interval should be programmable.)
(2) What were the lowest and highest levels at which a significant number of cloud occurrences were observed? (The number should be programmable.)
(3) How many cloud observations occurred below a specified critical height? (This height should be programmable.)

Additional Instruments

Other instruments which may be included are the following:

(1) Sky cover detector, for the fraction of sky obscured by clouds;
(2) Photoswitch, for indicating background light level;
(3) Weather vane, for wind direction;
(4) Weather element detectors, for snow, rain, hail, etc.

In addition, some quantities are set in as manually operated switches, such as obscuration type, whether snow or homogeneous fog; and, in the case of airports, runway and approach light settings.

ALCH and RVR

Two quantities which have not been covered in the discussion of individual instruments are Approach Light Contact Height (ALCH) and Runway Visual Range (RVR). These quantities are of importance in landing aircraft under conditions of reduced visibility, either because of ceiling conditions, or ground obscuration. ALCH is the height at which the approach lights will be visible to the pilot. Actually, a high and low value are displayed for ALCH, since there is a statistical uncertainty to this kind of data. The higher value is the height at which there is a 20% probability of seeing the lights; the lower value corresponds to a 90% probability. Since the lights may be obscured either by clouds or by the presence of fog or snow, a variety of inputs are required for the ALCH determination. The inputs are:

(1) cloud height data from the ceilometer,
(2) transmissivity, as indicated by the transmissometer,
(3) snow or homogeneous fog, as indicated by an observer,
(4) background illumination (day or night conditions) from the photoswitch,
(5) approach light intensity from the approach light switch setting.

If either the transmissometer or the ceilometer indicate that limiting conditions may be present, a determination of ALCH is made. If both conditions are present, the results are compared and the lowest value is displayed. The actual determination can be made either by table look-up or by calculation. The equation, however, is based on Allard's Law, which takes into account the complex manner in which the human eye responds to different light levels.

This is a transcendental equation, and a very fast computing speed is required because of the iterative nature of the solution. The ALCH value should be updated once per minute. The use of table look-up requires about 18 tables, each having about 90 three-digit numbers. The tables are determined more or less empirically, and vary from one location to another. It was concluded in the present case that table look-up would be preferable, using magnetic drum storage for these and other look-up tables. The RVR determination is similar to that for ALCH, but not quite so complex, in that only ground conditions need be considered, and only a single number need be prepared, rather than the two probability levels required for ALCH. The RVR determination uses the following inputs:

(1) transmissivity,
(2) background illumination (day or night conditions),
(3) runway light setting.

The expression for RVR, in terms of Allard's Law is given below, for purposes of illustration:

$$C = \frac{V_R}{2R} \text{ Log } T_R - \text{Log } V_R$$

C is a constant based upon prevailing conditions,
V_R is the desired RVR value, and ranges from 1000 feet to 6500 feet or greater,
T_R is the transmissometer reading,
R is the transmissometer baseline (generally 500 feet).

It can be seen that the above equation cannot be solved explicitly for V_R; hence, a lengthy calculating procedure would be required. For this reason, table look-up was chosen. About 9 tables are required for the RVR determination.

From a consideration of the various input devices, it is possible to compile a list of capabilities which are desirable in the input portion of the automatic station. These would include the following:

(1) sampling,
(2) counting,
(3) averaging,
(4) timing,
(5) comparing,
(6) analog-digital conversion,
(7) peak-value detection,
(8) contact sensing,
(9) code conversion.

Many of these operations could be done either by the input circuitry or by the data processor. The choice is determined by the relative convenience, and the time available. It is desirable, wherever possible, to receive data from the instruments in the simplest possible form. This simplifies the instrument and leaves the data processing to be performed within the computer, capitalizing on the advantages of digital operation. It has been demonstrated repeatedly that the central data processor portion of a computer complex is much more reliable than the associated peripheral equipment. Furthermore, the computer can be monitored, and repairs facilitated, through the use of test and diagnostic routines, whereas the instruments are often difficult to check and to repair.

Description of the AMOS IV System

It can be seen that the machine needed for the automatic weather station is highly specialized, with a number of unusual characteristics. The salient features are listed below:

(1) The machine must accommodate a number of input devices, all furnishing data continuously.
(2) Extensive stored tables are needed for empirically determined data which varies from station to station.
(3) A short word length is sufficient, since the data comes primarily from physical instruments; three digits and sign appear sufficient, relying on double-precision methods for those few cases where needed.
(4) A comparatively slow circuit speed is acceptable, working in conjunction with the magnetic drum, which rotates at a moderate speed for long life and reduced cost.
(5) The machine needs only a limited arithmetic capability, in view of the extensive stored tables; it can perform addition and subtraction, with other operations available through programming.
(6) The machine must transmit teletypewriter messages at high and low speeds, independently of each other and of the data processor.
(7) Provision must be included for operating local and remote displays.
(8) The machine must concurrently process input data, transmit teletypewriter messages, and perform data processing.

For purposes of discussion the machine may be analyzed in two sections: The input-output portion, and the central processor. The input-output circuitry is concerned with collecting the instrument outputs, pre-processing them where necessary, and making the data available to the processor. This portion also receives data which the processor has assembled in special output tracks and prepares the appropriate teletypewriter messages. The input-output circuitry is wired for specific tasks although considerable latitude has been left for modifications and additions. The central processor is internally programmed and is controlled by an automatic typewriter which also can be used as a form of display. A block diagram of the AMOS IV system appears in Fig. 1.

Input from Instruments

The method of receiving input data from the weather-sensing instruments is a compromise between the use of separate pre-processing devices and use of the central processor. In order to avoid excessive interruption of the central processor, varying amounts of circuitry have been assembled, depending on the form of the input data, to pre-digest the instrument signals for most efficient use by the processor. Once the data has been prepared in suitable form, generally as contact closures or storage in flip-flop registers, it is entered into the computer via an input-data track on the magnetic drum. This track is equipped with two heads, one addressable by the central processor and the other wired to the input circuitry. Since the track can store 100 words, there is an input capacity of 100 instrument readings, a quantity considerably in excess of present requirements. The address of each word identifies the reading, and the addresses therefore, are used to call out the appropriate subroutines when new data appears in the various word locations. The input devices are sampled sequentially by means of commutating pulses obtained from a decoding network attached to an address counter. It is possible with this scheme to sample any instrument within 1/30 second of the time that a desired reading is obtained. If readings were obtained at the rate of 30 per second, however, the central processor would quickly be overloaded; actually, it is sufficient to sample most instruments at intervals of once per minute or longer. The ceilometer is the most frequent with readings at 6 second intervals.

In addition to an address counter and decoding network for obtaining commutating pulses, the input circuit has a one-word shift register which serves as a buffer between the instruments and the input recording circuit. Data words from sampled instruments are inserted in the register by means of a parallel transfer, up to 13 bits at a time (three decimal digits and sign). The number representation need not be binary-coded decimal, since the computer can perform code conversion, if required.

Among the quantities requiring more extensive pre-processing, the transmissometer is illustrative. Here, a pulse rate of 0 to 4000 ppm is to be counted and expressed as a 3 digit number. New values should be available once per minute, so a one-minute time base is used. The incoming pulses are shaped and passed through a two-stage binary counter which reduces the rate

by a factor of four. Thus, the maximum (clear atmosphere) pulse rate will yield a one-minute count of 999, which can be expressed with one data word. (A gating circuit inhibits counts beyond 999 to prevent overflow, should a higher counting rate inadvertently occur).

The end of a one-minute sampling interval is indicated by a pulse from a timer which counts drum revolutions, since the drum is driven by a synchronous motor. Thus, the drum is always in a known position, and there is a minimum of delay in sampling the desired reading and transferring it to the drum. The counter is inhibited during the sampling and resetting interval. (One-third millisecond is required for sampling, 20 microseconds for settling time after reset. No significant information would be lost during this short time).

Utilization of Input Data

Utilization of input data by the central processor is handled by a "watchdog" routine. Whenever a subroutine is completed, the processor proceeds to scan the input channel, starting at the beginning. When an instrument reading is noted it is read into a register and the address is noted. The address is keyed to a subroutine which directs the handling of the new reading. It may simply be placed in memory for future use, or inserted into an output message channel. More generally, however, there are several operations including perhaps a few calculations. A key step in the subroutine is the insertion of a nonsense word which could never be a valid instrument reading, and will be disregarded during the next "watchdog" scan. This will permit the "watchdog" routine to pick up the next succeeding instrument reading in the input channel.

Teletypewriter Output

The teletypewriter outputs involve the buffering of data, which comes from the drum at a high rate, down to the desired message speed. In addition, data words must be reorganized into teletypewriter characters, including the addition of start and stop pulses, and the generation of space and sign characters. Two independent teletypewriter outputs are required, with different codes and message formats. The low-speed output is nominally 100 words per minute, while the high-speed output is in the range of 750 to 1500 words per minute. Several different message lengths are required at the higher speed, requiring that the circuitry be capable of skipping unwanted portions of the message. Since the messages are to be combinations of data prepared by the computer and alphanumeric remarks and text inserted by hand, several tracks have been allowed on the drum for this information. Certain tracks, addressable by the computer, contain the numerical data. Other tracks may be written into only from the automatic typewriter, and are used for the remarks. These are all dual-head tracks, with one set of heads being used to insert data, either from the processor or the typewriter, while the other set is used to read out the information.

The low speed circuitry is fairly straightforward, since there are several drum revolutions in the time required to transmit the contents of one machine word. Thus, it is only necessary to provide a one-word buffer register, which in turn transfers its contents through an encoding matrix to an output shift register. As soon as the buffer register is empty, an address circuit proceeds to watch for the next consecutive word from the drum, which is then read into the buffer. Each machine word, consisting of three decimal digits and sign, is transmitted as five characters: sign, three digits, and space. Remarks are stored as five-bit teletypewriter characters, two per machine word.

The high-speed circuitry is complicated by the fact that the contents of several machine words are transmitted during a single drum revolution. This was handled by spacing the data around the drum so that the next word would be available just as the last one was transmitted. To assure synchronism between the teletypewriter and the drum, a clock track on the drum is used to furnish the teletypewriter pulses.

It can be seen that the various dual-head tracks are the means by which simultaneous input, output and data processing functions are performed. In order that these various heads may handle data in a manner compatible with the central processor, it is necessary that each pair of heads be equally spaced. A second set of sync pulses, delayed by an amount equal to the spacing between heads, has been provided.

Data Processor Section

The data processor portion of the AMOS IV system is constructed as a separate entity which can be replaced at a later date, should a more powerful computer be required. The processor is built around a magnetic storage drum having 100 general storage channels and a number of dual head registers. The general storage channels contain stored instructions, subroutines, diagnostic routines, and look-up tables. The dual-head registers are used for the handling of input data from instruments and for output information to the displays and to the teletypewriter lines.

The processor uses a binary-coded decimal number representation with a word length of three decimal digits and sign. A parity bit is added for a memory check, giving a word length of 14 bits. There are 100 words per channel, making a total of 10,000 words, and this number can be expanded to 20,000 through the use of additional heads. The drum operates at a conservative rate of 1800 RPM; non-return-to-zero recording is used, with a recording density of 120 bits per inch. Thus, the machine operates at a bit rate of 50 kc.

The computer is an internally programmed, single-address machine with about 21 operations. The operations are listed in Table I. The operations have been grouped together and coded according to function for convenience in programming. The operations are generally quite similar to other small machines. The memory scan operations (60, 64, 69) are specifically desiged to optimize table look-up. Thus, table look-up may be performed with fewer instructions and with only one drum revolution. The index register should be called an alternative addressing method. When an address is placed in the index register by one of the memory scan operations, it will be transferred to the instruction register only when a non-decimal combination of bits appears in the word-select portion of the instruction register.

An instruction utilizes two consecutive words from memory, providing six decimal digits and two signs. This is shown in Fig. 2. Two digits are required for channel identification and two for the word location within a channel. In order to allow for ease in the programming of address modification, the word address uses the first two digits of the first word, while the channel location uses the second two digits of the second word. The operation code is divided between the two words, using the remaining two digits and the included sign.

The processor has three basic modes of operation. The normal mode allows the machine to automatically sequence through the instructions until a halt code is encountered in the instruction register. The machine halts on any take-in or print-out operation for which the sign of the instruction is negative. A breakpoint mode is used for program debugging or machine maintenance. In this mode, the machine will halt for any instruction having a negative sign. A single-step mode of operation is included, and is used primarily for machine diagnosis or program debugging. In single-step operation the machine performs a single operation and halts. A complete operation consists of executing an instruction and bringing the next instruction from memory. All the registers and major portions of the machine are displayed on an indicating panel.

An automatic typewriter with punched paper tape is used as the primary input-output means for the processor. In addition to its use with the processor, the typewriter may be used off-line, for the preparation and verification of punched paper tapes.

Construction

The machine is constructed from a series of transistorized building blocks previously developed at the National Bureau of Standards. Three factors were emphasized as the main considerations in designing these packages: reliability, cost, and versatility.

Reliability

(1) The circuits are designed to permit wide variations from the nominal values of the characteristics and parameters of the components.

(2) The electrical outputs from most of the packages can be short-circuited to ground or to the negative voltage supply without damage to any of the components.

(3) Pin-type connectors with high-pressure contacts are used rather than printed-circuit edge-type connectors.

(4) Signal swings are at least 6 volts, with a collector supply of -12 volts.

(5) All connectors have gold-plated pins.

(6) All back panel wiring is by taper pins for ease and convenience in making external connections. Taper pins also eliminate solder joints.

Economy

(1) "Entertainment"-type germanium transistors are used throughout the circuitry.

(2) The wide tolerances permit using unselected "off-the-shelf" components. However, transistors and diodes are tested for open or short circuits before assembly into the packages.

(3) All connections, including those to the connector, are made by dip-soldering the board.

Versatility

(1) Two triggering gates are included with each flip-flop circuit to permit connecting the package as either a counter or a register without having to use additional gates from some other package.

(2) The bases of the transistors on flip-flop circuits are accessible at the connector so that an unlimited number of additional input gates can be connected.

These packages have been in use over a period of several years in a number of units of laboratory equipment. The high reliability which they have demonstrated has been very gratifying.

Approximately 500 of these packages, with an average of three transistors each, are contained in the central processor portion, with about 400 additional packages in the input-output portion. The packages are contained in drawers which in turn are mounted on slides, permitting practically all maintenance to be accomplished from the front.

Conclusion

It should be emphasized that the AMOS IV System is intended as a research tool for exploring the use of automatic data processing in the handling of weather data. Emphasis has been placed on versatility, in view of constantly changing meteorological requirements. It is felt that the experience gained from the use of a limited number of AMOS IV systems should permit the formulation of much more realistic designs and specifications for future automatic weather stations based on the use of automatic data processors.

TABLE 1

AMOS IV Data Processor Instructions

Code	Operation
00	Take in ten words from typewriter starting with memory location α β.
02	Take in two words from typewriter starting with memory location α β.
09	Take in full channel of words from typewriter into memory channel β.
10	Print or punch out ten words from memory starting with α β.
12	Print or punch out two words from memory starting with α β.
19	Print or punch out full channel of words from memory channel β.
21	Read one word from memory location α β and place into A register.
22	Read one word from memory location α β and place into B register.
31	Take word from A register and place into memory location α β.
32	Take word from B register and place into memory location α β.
41	Add contents of A register to contents of memory location α β and place answer into B register.
42	Subtract the contents of memory location α β from A register and place answer into B register.
45	Shift contents of A register right 4 bits.
50	If B = 0, jump to contents of α β for next instruction.
51	If overflow occurs, jump to contents of α β for next instruction.
59	If B < 0, jump to contents of α β for next instruction.
	Memory scan instructions. Contents of A register are compared with words in channel β.
60	When $(\alpha_A\beta)$ = (A), place α_A in I register; stop at first occurrence.
64	When $(\alpha_A\beta)$ > (A), place α_A in I register; stop at first occurence.
69	Seek largest word in channel β; place α portion of address in I register.
73	Transfer bits described in β from register B to register A.
93	Read out three words from memory starting with location α β into output display registers.

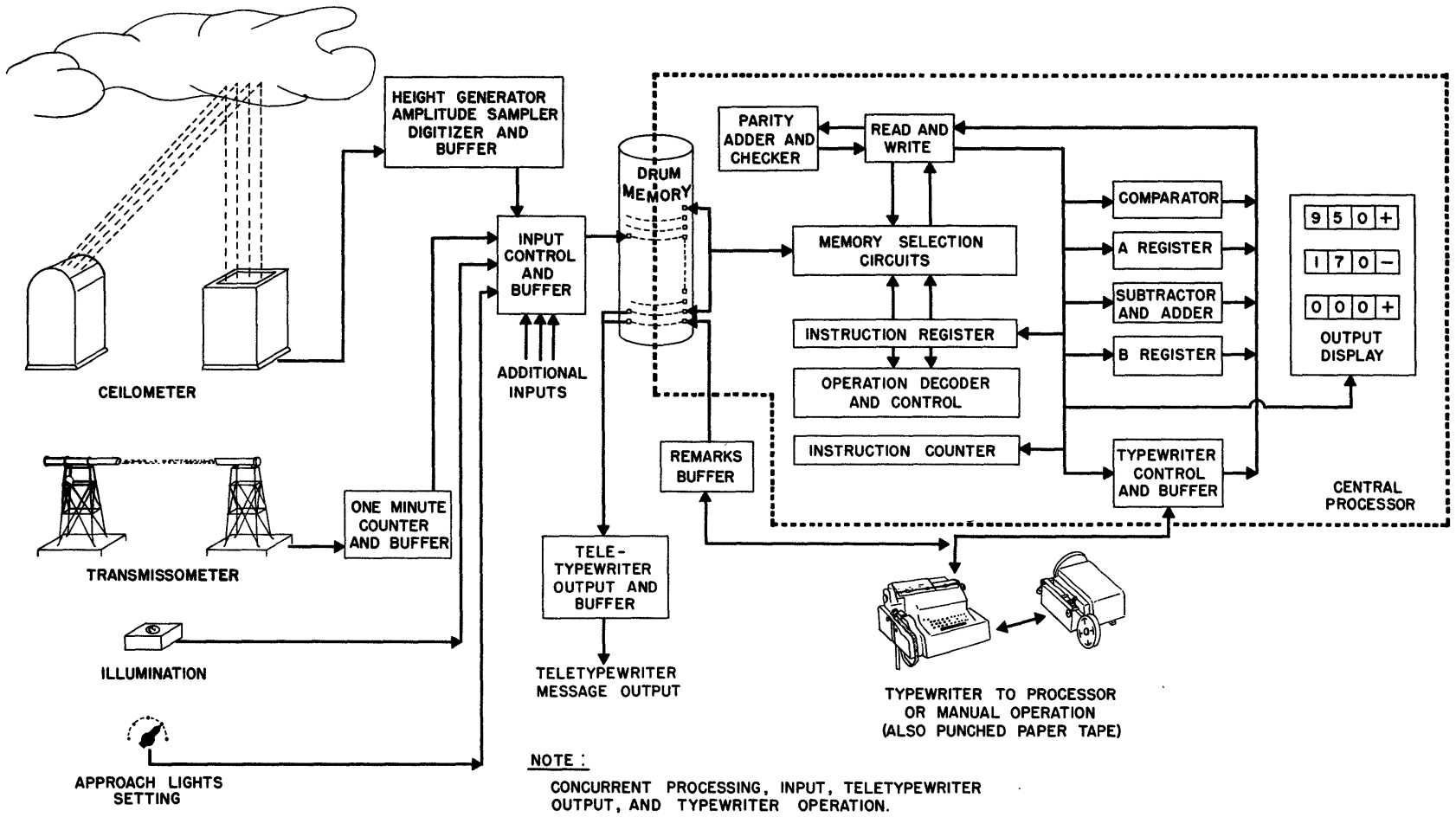
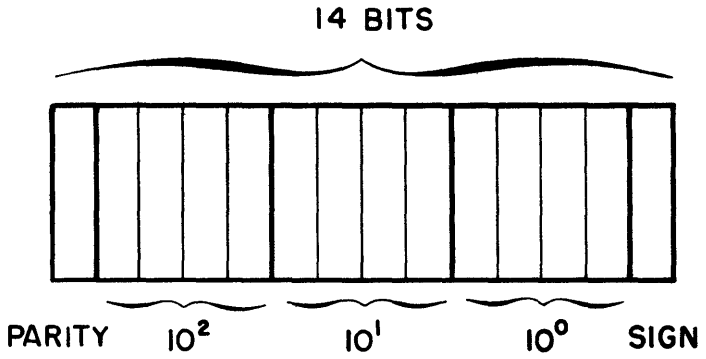


Figure I. Block Diagram of AMOS IV With Approach Visibility Configuration

DATA WORD



INSTRUCTION

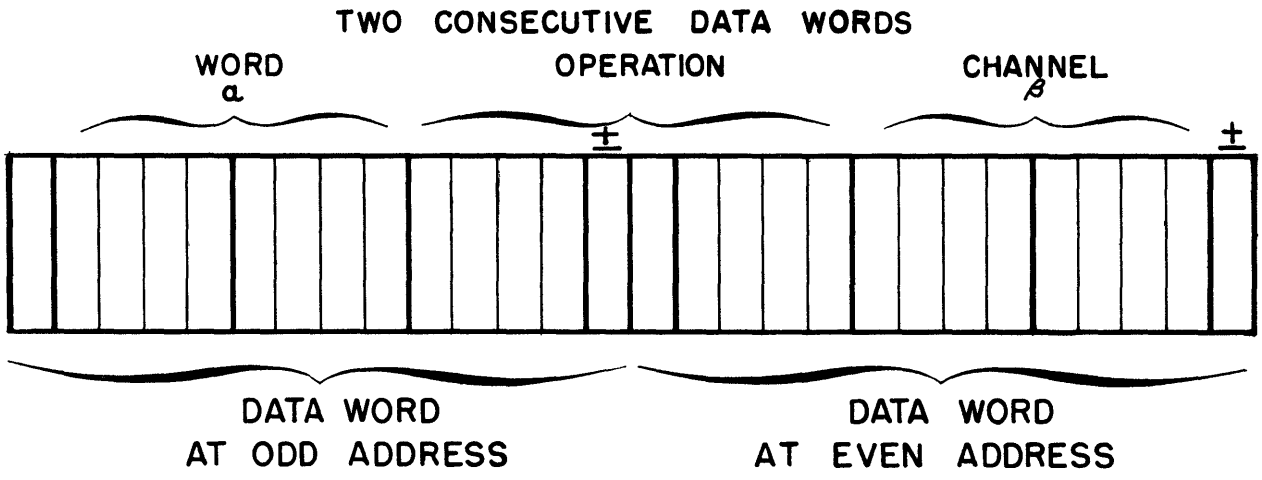


Figure 2. Word Format of the AMOS IV Computer.

A SURVEY OF DIGITAL METHODS FOR RADAR DATA PROCESSING

F. H. Krantz and W. D. Murray

Burroughs Laboratories
Paoli, Penna.

Summary

This paper reviews the growing number of declassified techniques for automatic processing of radar data by digital means. Emphasis is placed upon signal time-sampling and quantization, integration methods, rejection of stationary targets, radar trigger manipulation, and treatment of radar beacon code data. These techniques are discussed individually and are also shown combined in a hypothetical radar data processor design.

Introduction

Radar, a new technology of World War II, has become an almost common part of our modern life but continues to be the subject of new applications, new methodology and new techniques. In its first decade, the radar system consisted of the radar itself, a cathode ray tube display and a human operator. As the technology matured and radar targets gained new capability, the need for extracting more information from the radar signal at a more rapid rate grew. Thus, in the past decade, the increased performance of the radar has been accompanied by the growth of the associated field of radar data processing.

The first radar data processors used analog techniques in an attempt to automatically reproduce those operations performed by the human. More recently, the pressure for increased automation, coupled with the requirement for automatic communication of radar information over long distances, has resulted in the introduction of digital techniques for radar data processing. It is the purpose of this paper to outline the principles of digital radar data processing and to demonstrate examples of application of digital techniques to this field.

The expert in digital computer technology will immediately recognize most of the digital mechanizations and will observe that in radar data processing these usual digital techniques are applied, sometimes in completely different form, to this new problem area. It should be noted that most of the applications described herein, while shown for the radar problem, are equally applicable to other problems in extraction of information from a signal in presence of noise. Typical are the fields of SONAR, infrared detection, magnetic detection and communications.

The Basic Radar Problem

In general terms, the radar data processing problem is one of information extraction; that is, it is desired to extract from the radar signal the maximum amount of real information and at the same time to exclude extraneous information introduced by noise and other target-like phenomena. For the purpose of the present discussion, the simplified pulse radar system of figure 1 will be the vehicle to which radar data processing is applied.

The pulse radar consists of a transmitter which periodically transmits a burst of energy of prescribed pulse duration at a prescribed carrier frequency. This burst of energy will strike targets and some portion will be returned in the direction of the radar equipment. The character of the returning signal will have been modified by the addition of a doppler frequency component proportional to target velocity.

The amplitude of the returned signal is a function of effective cross sectional area of the target. On successive radar returns this amplitude might vary due to changes in cross sectional area.

In addition to target returns there will be returns due to clutter caused by precipitation, by objects on the ground, and by such effects as aurora. Each of these clutter-reproducing objects will impose its own effective doppler frequency on the signal and, in general, these doppler frequencies are lower than those for

the usual real target. The real target is further obliterated by effects of noise in the radar receiver or by interference generated by some other electronic equipment, either hostile or friendly. The noise and many of the interfering signals have random frequency and amplitude distributions, a characteristic frequently used in radar data processing for their elimination.

In the radar receiver the frequency of the signal is reduced from that of the basic carrier to some intermediate frequency by comparison with a reference oscillator. Information on amplitude of the returning signal (the envelope of the IF signal) is presented at the output of an amplitude detector as "normal" video. Range of the target is represented as the time of occurrence of the pulse. Although much of the radar data processing is applied to the normal video signal, the phase information contained in the IF is also useful in evaluation of target velocity. We will not be concerned here with the high frequency characteristics of the returning pulse which are used for more sophisticated forms of discrimination based on target signature.

From the information received from the radar equipment, it is necessary to extract information on real targets. The general objective is to obtain through radar data processing the maximum sensitivity to weak targets while maintaining a minimum rate of generation of false targets. The characteristics of a real target which are different from those of noise and clutter are used in the processing of the radar data. Among these are that a real target will correlate in range position and in radial velocity from pulse to pulse during the time that the radar illuminates the target, and that real targets will, in general, have a greater doppler velocity than will signals due to clutter.

Analog Detection Based on Signal Amplitude

In the early pulsed radar systems information was processed by presentation on a cathode ray tube display to a human observer. The typical display will sweep an electron beam from the center of the display at an angle equal to that of the radar antenna at a uniform speed and at a sweep repetition rate coincident with the radar prf. The beam is intensity modulated so that the light output on the surface of the display will be proportional to amplitude of the raw video signal.

A real target will appear at the same range position for several successive radar pulses, and thus, will appear as a short arc on the display. Random noise, which does not correlate in range, will generally appear as isolated spots randomly distributed over the surface of the display. These separate characteristics of targets and noise are used to discriminate between the two. In the simple radar system, the integration is performed in the memory of the human operator assisted by the relatively long persistence of the display phosphor. This method is quite sufficient for the separation of strong targets and weak noise, particularly when the observers attention need only be devoted to one or two targets on the face of the display.

Electronic techniques are sometimes used to augment the integration process in a device known as the video integrator. This device consists of one or more delay lines, each adjusted to the radar pulse period. The outputs of delay lines, each modified by a suitable decay factor, are added and their sum is used to intensity-modulate the PPI display. A typical video integrator is shown in figure 2. Although the video integrator increases the capability to discriminate between targets and noise, its characteristics of essentially exponential decay are not completely matched to the energy distribution across a radar beam width. Thus, the optimum discrimination between target and noise signals is not obtained.

A typical display from a simple radar system is shown in figure 3. The problems of extracting information on a large number of targets to high accuracy in the presence of noise are readily apparent, and the need for more advanced processing techniques is demonstrated.

Digital Detection Based on Signal Amplitude

Digital methods for radar data processing start by quantizing the analog raw video signal in amplitude and range. As will be demonstrated later, it is possible to quantize in amplitude to as many as 8-bits, but for simplicity in demonstration of digital detection techniques, simple 1-bit encoding will be considered here. The inputs to the quantizer (figure 4) are the raw video from the radar receiver and a range timing reference generated by a digital clock. An output pulse is transmitted whenever the input signal is of amplitude larger than a fixed or automatically adjusted threshold (clip level) at

the time of occurrence of a range reference pulse.

The clip level is established by consideration of sensitivity required and false alarm rate reduction possible through later statistical integration processes. For the typical ground-based, long-range radar equipment, the clip level is established at an amplitude such that at 10% of the range timing pulses an output pulse will be generated. This very low clip level, permits detection of targets of very small radar cross section at extremely long ranges. The automatic adjustment of clip level based on long time sampling of the quantizer output is used to compensate for variations in gain of amplifiers in the radar equipment and in the quantizer.

The range reference clock frequency selection is based on requirements for range accuracy and range resolution in the radar. In order to provide maximum sensitivity and maximum resolution consistent with duration of the radar pulse, a range reference period equal to the pulse duration is typical. Thus, range resolution and accuracy of from 1/4 mile to several miles are found in the usual systems.

Following the quantizing process statistical detection is performed to eliminate signals due to noise while retaining those caused by targets. The simplest digital integrator, the exponential detector shown in figure 5, operates exactly as does the video integrator described above. In each range increment a target history is maintained by adding the new signal (either 1 or 0) to the sum of the previous signals multiplied by a constant less than unity. The size of the constant is determined by azimuth resolution and target sensitivity objectives. When the sum contained in a register exceeds a preselected threshold, detection criteria have been met and an output indicates presence of a real target at that range. This indication is maintained until the "end target" threshold is greater than the register sum. The growth of the signal output of the exponential detector and the decay of such output after the radar beam has passed through the target for the case of a radar output of square characteristics are also shown in figure 5. The non-symmetry of the output signal and the mismatch with the radar beam make this type automatic detector less than optimum in detection probability, noise rejection and azimuth determination.

The more advanced sliding window detector increases detection sensitivity and azimuth determining accuracy. In this detector demonstrated in figure 6, a record of target return is maintained for each radar pulse through the radar beam width. The outputs of each pulse position of the record are summed and compared with a detection threshold and when this threshold is exceeded, start of target is called. At a later time when the sum of target histories in the radar beam width is less than the detection threshold, end of target is called. The symmetry of the output of this device and the match with the ideal radar beam are shown in figure 6.

Although the sliding window detector offers improved performance over the exponential detector, the equipment required in its implementation for a radar with many transmitted pulses per beam width is significantly larger.

In order to obtain performance approaching that of the sliding window detector, with equipment more closely approximating the exponential detector, a new statistical detector known as the moving sum detector has been developed. In this detector, (figure 7) a feedback loop is used to add the latest target amplitude to those of previous targets as in the exponential detector but the decrement is taken as the average amplitude over N main bangs. N may be as long as the integrating period or may be less in order to achieve maximum azimuth accuracy.

All of these automatic detectors require the use of memory in order that target history can be maintained. These memories are usually range-organized. That is, each range interval is represented by a prescribed memory word and at completion of processing of a detected target, the position in memory of such detected target establishes its range. Although ferrite core memories are also used, the range organized memory can be most easily demonstrated as implemented on a drum surface as shown in figure 8. Here a number of individual drum channels (or bits in each memory word) are used for the data processing function. The drum rotation rate is automatically coupled to the radar pulse repetition frequency; thus any radar range is represented by one position (or one word location) on the drum surface. A recirculating drum system with a positive erase bar is usually used in this application.

History of target signals is maintained by reading information from memory heads spaced by one pulse period from the write heads by writing back the previous history continuously as new target signals are introduced. In the sliding window detector, for example, information read from detector channel No. 1 is written

back into detector channel No. 2, from No. 2 into No. 3, etc. Thus, at any instant of time, information appearing under the read heads is a target history for that range interval over the prescribed number of previous radar pulses.

Azimuth Determination

To determine azimuth of detected targets a binary representation of antenna direction must be maintained. Although this can be done using a binary code wheel device in the antenna pedestal, it is more usual to have a pulse transducer, with each pulse representing an increment of antenna rotation, feed an azimuth counter which is cleared to zero at the time the antenna passes through a north reference position. Thus, the azimuth counter maintains a continuous record of antenna orientation. The detector operating curves of figures 5, 6, 7 show that the center of a target is represented by the average of the azimuth at start of target and end of target corrected for some offset introduced by the detector.

A typical method for accomplishing azimuth determination is to add one to a sum maintained in the azimuth channels of the memory (figure 8) whenever the statistical detector output is greater than the detection threshold in each range increment. At "end of target" 1/2 of this sum is subtracted from the number in the azimuth counter. The azimuth offset correction, which is fixed for the statistical detector used, is then applied to obtain an accurate digital representation of target azimuth. It is in this process that the symmetry of the detector output is important in obtaining azimuth accuracy.

In the computation of target azimuth by the method described above, information on target strength or azimuthal extent is automatically available. This target characteristic, known also as "run length," is of importance in separating real targets from targets produced by clutter.

Target Buffer Memory

In cases where automatic communication of target information is required, problems may be imposed by bandwidth limitations of communications equipment. Since the data processor described thus far operates in real radar time, it is possible for targets to be detected at a rate much greater than the average rate over a radar rotation. It is not economically practical to build a communications system designed to operate at the maximum rate and a queuing problem is introduced. If a target buffer memory is included in the radar data processor, it is possible to design the communications system with an information rate equivalent to the average target detection rate and still minimize the loss of target information due to communication line saturation. This memory may be provided as a part of the statistical detector memory wherein target storage locations are included in each range interval or through a separate buffer memory where targets are inserted as they are detected and are removed as space is available in the communication facility. Although the first method is wasteful of memory space in that a large number of unneeded memory locations are provided, it is efficient in a drum memory where separate drum tracks are relatively inexpensive when added to the basic drum required for the detection and azimuth calculation process. In the case of a ferrite core detector memory, however, memory cost goes up more rapidly with an increase in memory location requirements and it is often more economical to add a small independent random access buffer memory. For a typical ground radar environment, where probability of loss of targets due to saturation must be minimized, the radar data processor will include buffer memory for two targets per range interval in the case of the first type of buffer or will include 25 or 50 memory locations in the case of the independent buffer memory.

PRF Control

In many applications, devices are required to selectively adjust the period between radar triggers in order to achieve certain desirable performance characteristics. Among the advantages obtained are interference rejection, extension of the basic radar range, and elimination of loss in sensitivity in processing of the target doppler frequency shift. Each of these is briefly discussed below.

When a radar operates in the vicinity of other similar radars, as frequently occurs in military zones, it may happen that pulses from a neighboring radar are received in one's own radar and appear as point targets. When this situation is particularly aggravated, dense spirals are formed on the radar presentation and considerable excess data is created. Periodic or random variation in the radar trigger of one's own radar, when followed by range realignment and integration of the video, will

effectively eliminate this form of interference.

In a constant PRF radar the basic range of the radar is limited by the distance which a pulse can travel and return in the time between radar triggers. By coding or otherwise manipulating the interpulse periods so as to achieve various pulse separations, echoes from beyond the basic range of the radar can be recognized and detected without range ambiguity.

Various techniques for the elimination of stationary objects while retaining moving targets are totally dependent upon the doppler shift in the echo from the moving object to carry out the required discrimination. In a pulsed system such as radar, however, targets traveling at a speed such as to move one r-f wave length between pulses will be indistinguishable from stationary objects. Controlled variation of the radar intra-pulse period will permit recognition of moving targets under all conditions without sacrifice in the ability to reject stationary objects and clutter.

In practice, the equipment needed for PRF control and the subsequent manipulation of target data is particularly simple and straightforward when carried out with digital methods. The simplest technique for control of the intra-pulse period is to make use of a feedback counter. Using this logical configuration to select various delay lines to be inserted in the trigger generation circuitry, a number of values of delay are readily obtained. A complete system is shown in figure 9.

Methods for realignment of the radar echoes in range for integration and other purposes depend upon the type of memory employed. In systems utilizing a magnetic core memory, no special provisions need be made other than synchronizing the flow of data through the memory to the radar trigger. Where the memory is a magnetic drum, servoed to the average PRF, the drum inertia prohibits rapid adjustment of the drum speed to follow the changes in interpulse period so that complementary delay lines in the receiver chain are needed to realign the receiver video. The conversion of video to digital form prior to range realignment achieves a number of desirable economies in the engineering of the complementary delays.

Clutter Rejection

In many radar installations the presence of large distributed reflectors such as land masses and cloud banks will give rise to a very large number of returns which obscure the radar picture for the manual observer and create very objectional quantities of excess data in automatic data processing systems. (See figure 3.) A variety of techniques have been studied which have as their objective the elimination of these clutter returns without degradation to the system sensitivity for moving targets. Many of these solutions have been only partially successful.

An early technique having as its primary objective the control of excess data utilized an operator to manually map out the clutter areas. One equipment for this purpose (shown in figure 10) consisted of a Plan-Position Indicator (PPI) display over which was suspended a photomultiplier tube sensitive to the ultra-violet layer of the PPI phosphor. Mapping is accomplished by the application of an opaque inking fluid to the clutter areas to be removed. Although this system was particularly simple to implement using digital techniques, all true targets within the map area were eliminated with the clutter.

A second fairly straightforward technique eliminates clutter on the basis of its lower relative strength. One approach uses as a measure of strength the instantaneous amplitude of the video return and, by comparison with a preset reference level, rejects all signals of less than the reference strength. The reference level is selected on the basis of empirical data to permit strong point targets to be passed by the comparator circuit.

A second approach determines target strength on the basis of number of returns received from the target as the radar beam sweeps across it. The returns are counted and on the basis of prior statistical data, targets exhibiting too few or too many hits are rejected, leaving only those targets which have been previously shown to be point targets. This method requires the implementation of counters in each range element, but this is particularly simple when implemented in conjunction with the circuitry for azimuth estimation.

A third technique is essentially a refinement of the one just previously mentioned in that total number of hits received from a unit area are counted and used to set the system sensitivity level. By this means the detection level can be made just slightly greater than the instantaneous background so that all clutter is effectively eliminated. Targets in the clear or targets stronger than the clutter background are accepted by the system. This technique can be most easily implemented digitally as a part of any of the digital sweep integrators previously described.

A fourth approach to clutter rejection and the only one giving detection capability for targets weaker than the clutter utilizes the difference in velocity between clutter and targets to achieve the required discrimination. This system, frequently called Moving Target Indication or MTI, makes use of the doppler shift in the reflected energy created by the moving target and was until recently implemented by analog techniques. Digital methods, however, have been applied to this problem, resulting in elimination of bulky delay lines and achieving high reliability, freedom from field adjustment, and greater flexibility, while preserving equivalent performance. Modifications of the digital technique can be made to measure radial velocity on each detected target.

The digital approach to MTI is shown in figure 11. This simplified diagram shows a system for sensing either the x or y component of the target vector, means for encoding this quantity to a number of binary digits, a small memory to delay this data one inter-pulse period, and a digital subtractor; the purpose of the system is to compare successive values of the target amplitude or its components. Moving targets will exhibit a periodic variation in their amplitude components at a frequency given by the equation for doppler shift. These targets will, therefore, produce a non-zero difference between successive echo amplitudes and in subsequent circuitry will produce a detectable data processor output. Stationary targets having no periodic variation in amplitude will produce a theoretically zero output from the subtractor and will, therefore, be eliminated from further data processing. Systems of this general form have the ability to detect moving targets whose return echo strength is over 40 db weaker than the surrounding clutter.

Video Amplitude Encoding

Several special situations may occur in radar data processing where high speed encoding of video amplitude to high accuracy is required. The digital MTI system previously described is one such case; other cases arise in conjunction with stacked-beam radars which measure range, azimuth and height on each detected target. The height output is conventionally an analog voltage which must be encoded with great accuracy and high speed for further data processing. Encoding may also be required in those cases where only a few hits are available on each target, dictating maximum retention of all amplitude information in the integration process.

Encoders for these purposes can be designed to achieve an encoding capability of eight bits in three to four microseconds. One encoder implementation which has proven successful depends upon comparison of the analog video signal with sixteen reference levels, subtraction of the largest number of integral levels possible from the signal while maintaining a net positive balance and a second comparison of the residue with a second set of sixteen reference levels. This encoder has demonstrated the capability cited above and is free from short and long term drifts and environmental limitations.

Beacon Code Detection

Air Defense and more recently air traffic control are both heavily dependent upon means auxiliary to the radar for the precise identification of targets. Toward this end there has been very considerable development of cooperative devices carried in aircraft which, when interrogated by a special transmitter associated with the primary radar, respond with a distinctive identifying code. Detection, verification and interpretation of the received code train is most successfully carried out through digital techniques.

The principle digital components of this system, (called beacon or secondary radar), are timing circuits to generate various special pulse patterns for interrogation, and code-train processing circuits which a) eliminate spurious replies; b) achieve correlation of the beacon data with the radar echo from the same target; and c) interpret the code train for parity errors and for special codes used to indicate emergency, special aircraft, etc. The beacon data processing system is typically composed of high fidelity delay lines having a multiplicity of taps and conventional computer elements operating at relatively high speeds. (See figure 12.) Equipment currently under development to meet the exacting identification requirements of air traffic control may have as many as one hundred thousand electronic components.

Complete System Design

A hypothetical radar and its data processing system can be constructed to show the application of digital techniques in a fairly standard application. The radar transmitter chain is shown in figure 13. In this figure, the basic transmitter pulse repetition frequency is determined in the

trigger pulse generator on the left. Delays under the control of the PRF jitter network produce variations in radar inter-pulse period which are predictable but which will not repeat for an arbitrarily large number of radar trigger periods. The jitter pulses are then applied to a modulator which fires the final stage of the radar transmitter producing pulses of high frequency energy. A beacon transmitter chain produces coded interrogation pulses for transmission by a second antenna mounted upon the primary radar.

The receiving and data processing systems are shown in figures 14 and 15. At the upper left the incoming radar information is divided between two channels - one the so-called normal channel and the other the MTI channel. In the normal channel the radio frequency signal is first converted to an intermediate frequency, then to video, and then quantized to produce standardized ONES or ZEROS depending upon whether the target return is greater or less than a reference level automatically set in the digital quantizer. In the MTI channel the RF information is converted to an intermediate frequency, then applied to a phase detector which has an an output either the x or y component of the target vector. This component is encoded, delayed and applied to a digital subtraction network as previously described for a digital MTI. The subtractor output is reduced to a one-bit code and is then in all respects identical to the output of the Normal quantizer.

Simultaneously with the reception of target information from the prime radar, the response of the target to beacon interrogation is also received. After suitable demodulation from radio frequency through intermediate frequency to video, the code train is standardized in amplitude and timing, integrated to remove spurious replies, checked for parity, and caused to produce a signal which is additively mixed with the radar video to achieve a high detection probability at the integrator output. Suitable timing of the beacon trigger with respect to the radar trigger insures registration of the radar return and beacon response from the same target.

As shown in figure 15, the Normal and MTI signals are then gated on a range basis so that the MTI signal is selected for short ranges where clutter is prevalent and the normal video, exhibiting a slightly higher sensitivity, is selected for the longer radar ranges. The resulting signal is then passed to the binary sweep integrator which effectively eliminates false alarms produced by receiver noise and enhances system sensitivity. The integrator output actuates a beam-splitting logic which finds the center of the set of returns received from a point target as it is swept by the radar beam. The center is encoded by gating an azimuth counter to achieve a digital word for target azimuth. Range information on the target is readily obtained from the memory by gating a range counter on the basis of the position of the target in the range-organized memory.

Information on target range and azimuth taken from the prime radar and target identity taken from the beacon response are passed with other information on special target characteristics to a buffer memory to await transmission to the next user. The transmission media may be any low bandwidth system of which ordinary telephone lines are typical. Digital words, one per target, may be transmitted at rates in the order of 25 to 50 targets per second. Word-forming logic between the buffer store and the output section of the data processor arranges the range, azimuth, identity, and other information in prescribed order, carries out parity checks and generates the timing waveforms for actuation of the output function.

Summary

In summary, a variety of digital techniques have been very briefly described to indicate the breadth of the application of digital methods to radar data processing. While many specific techniques remain under military classification, the general principles of digital implementation of data processing functions are well understood and have in a number of instances given dramatic proof of their high reliability, ease of maintenance, and adaptability to modification and evolution, and in many cases provide functions and performance which have no counterpart in analog circuit technology. It is anticipated that with an increase in dependence on radar by both military and civil aviation, the number of installations of digital data processors will increase many fold and that the art of digital processing of radar data will steadily advance toward the solution of many of the complex problems still confronting the radar designer.

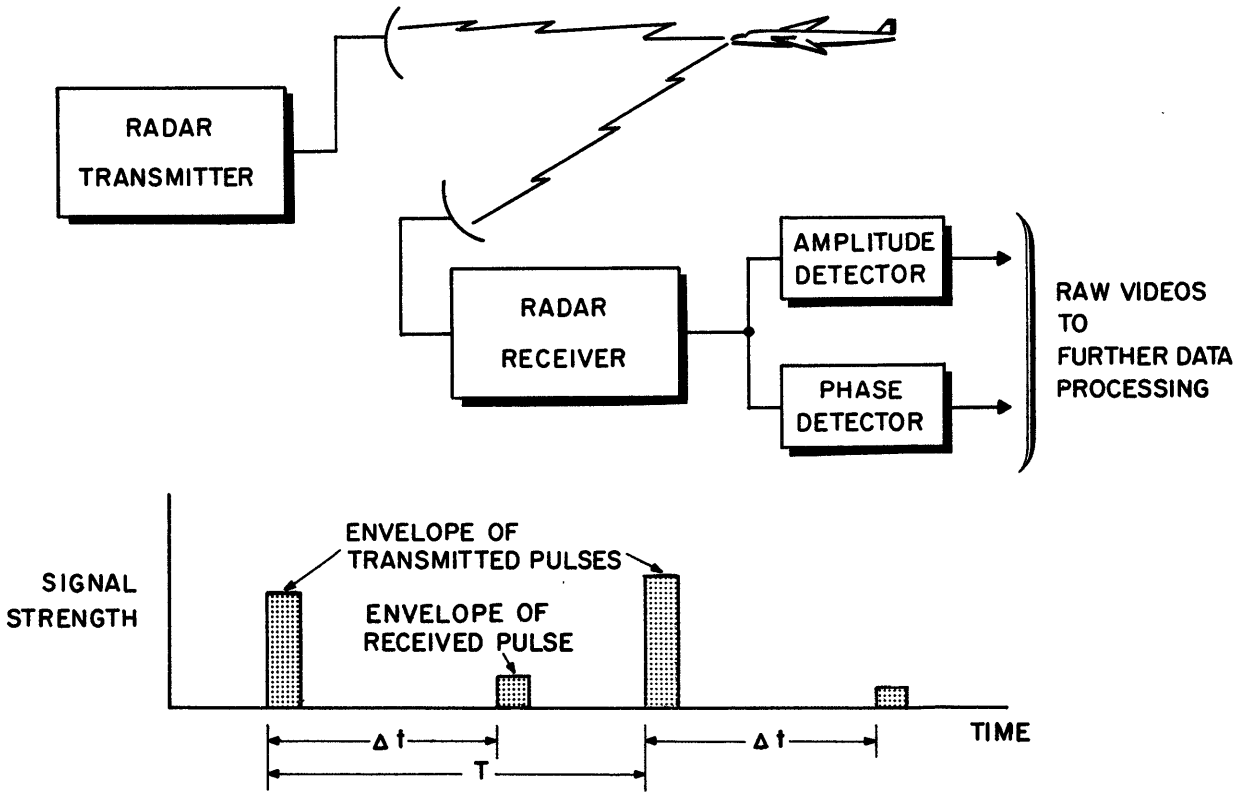


Fig. 1. Basic Radar System.

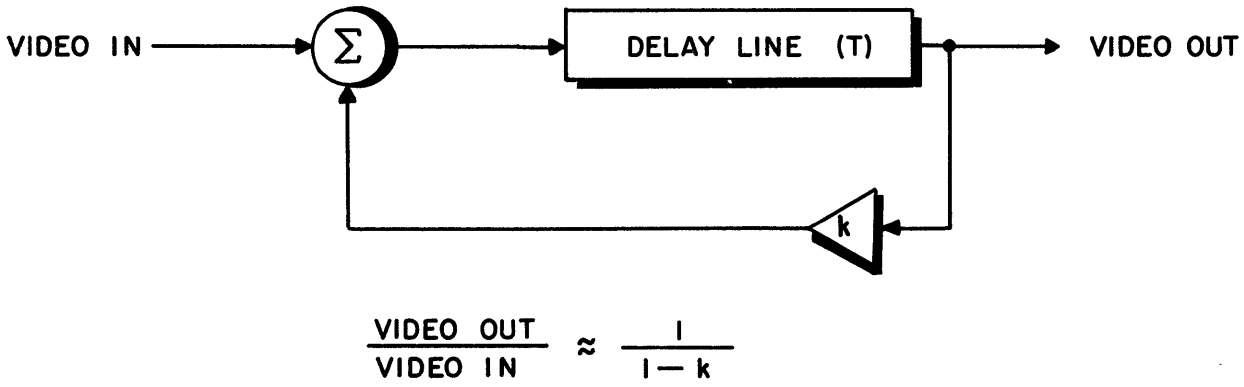


Fig. 2. Simplified Video Sweep Integrator.

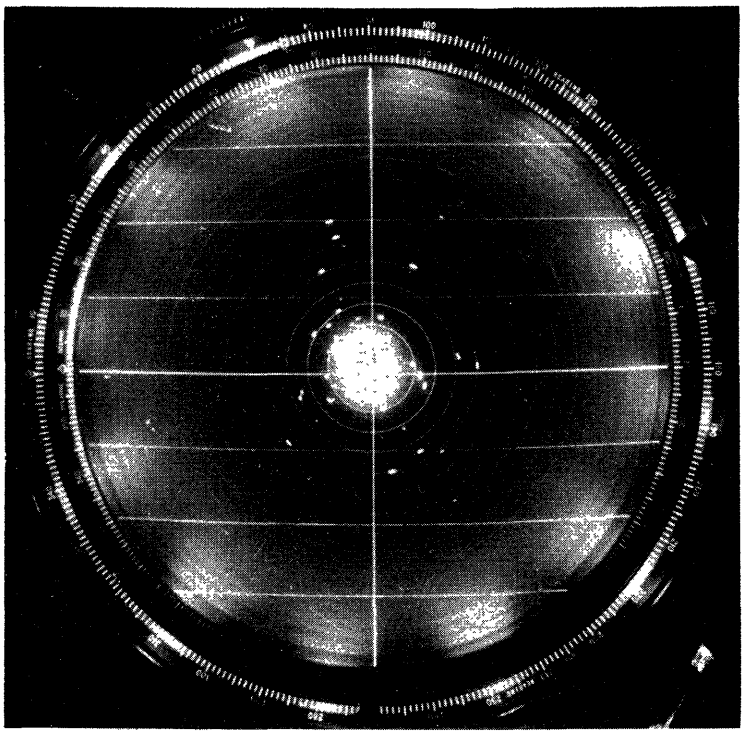


Fig. 3. Typical PPI Display.

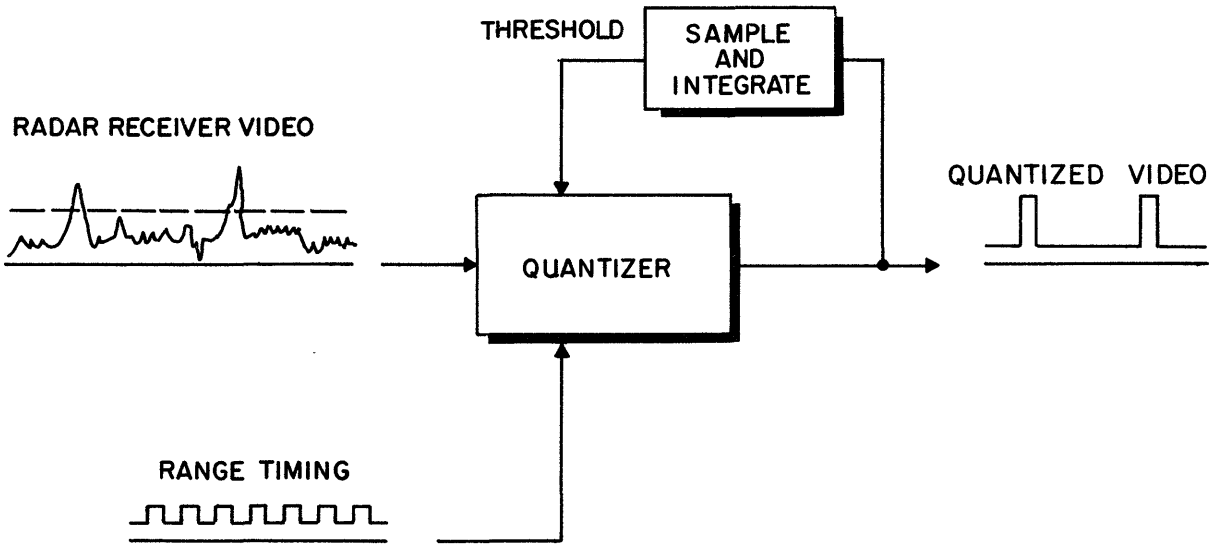


Fig. 4. Digital Quantizer for Radar Video.

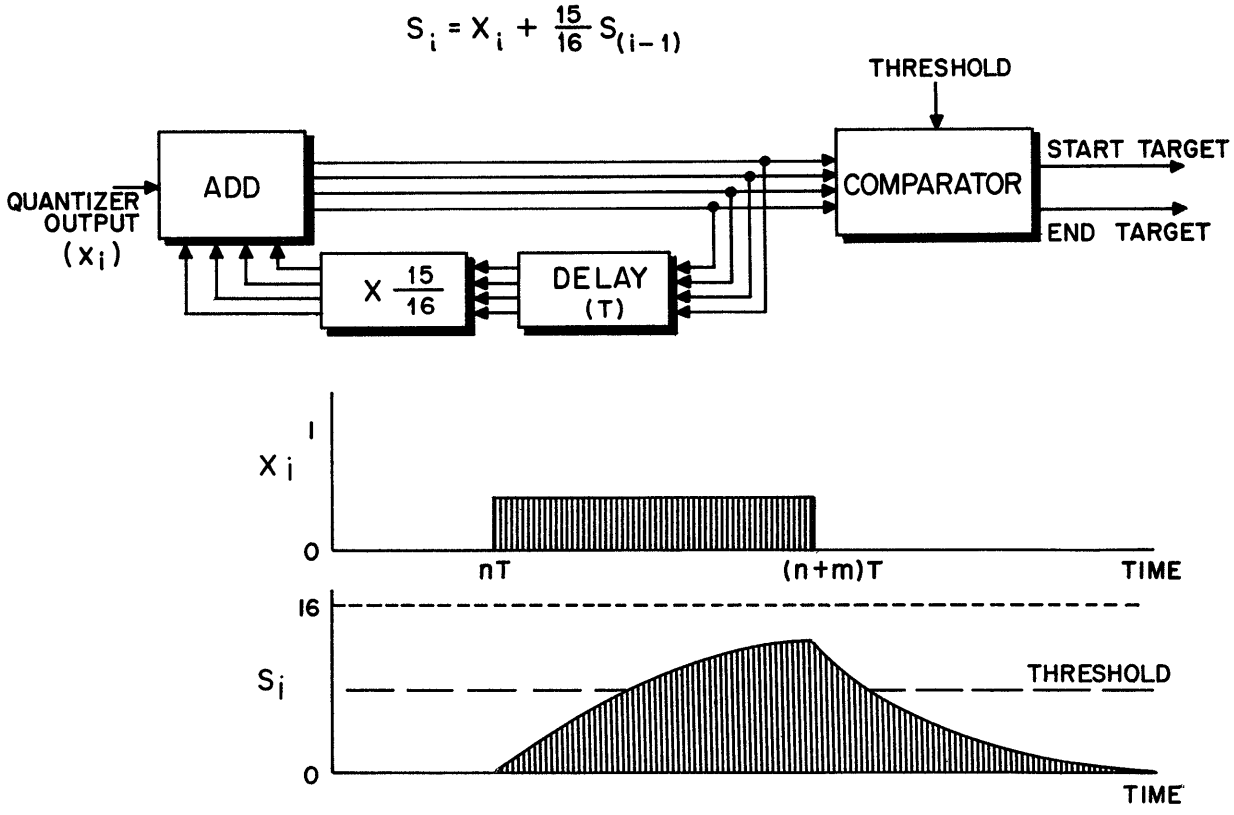


Fig. 5. 16-Hit Digital Sliding-Window Integrator.

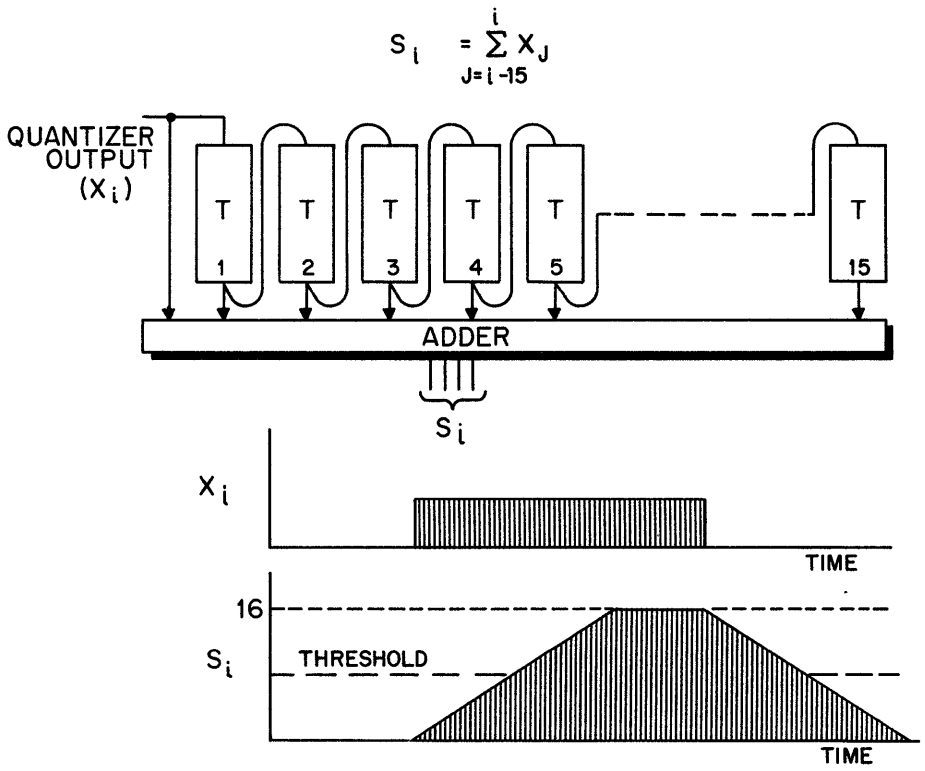


Fig. 6. 16-Hit Digital Sliding-Window Integrator.

$$S_I = X_i + S_{(i-1)} - \frac{1}{16} S_N$$

$$S_N = \sum_{J=N-15}^{N} X_J$$

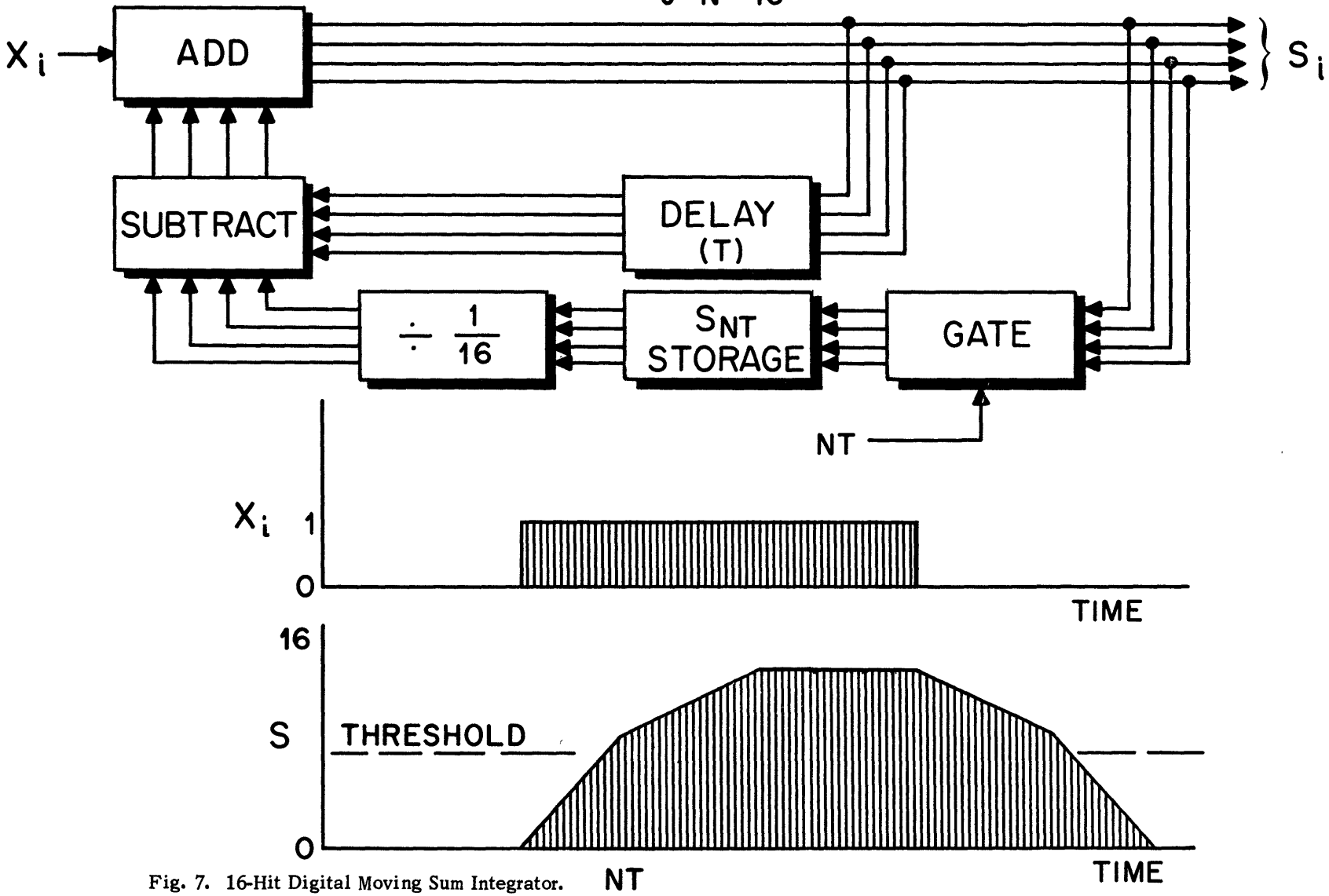


Fig. 7. 16-Hit Digital Moving Sum Integrator.

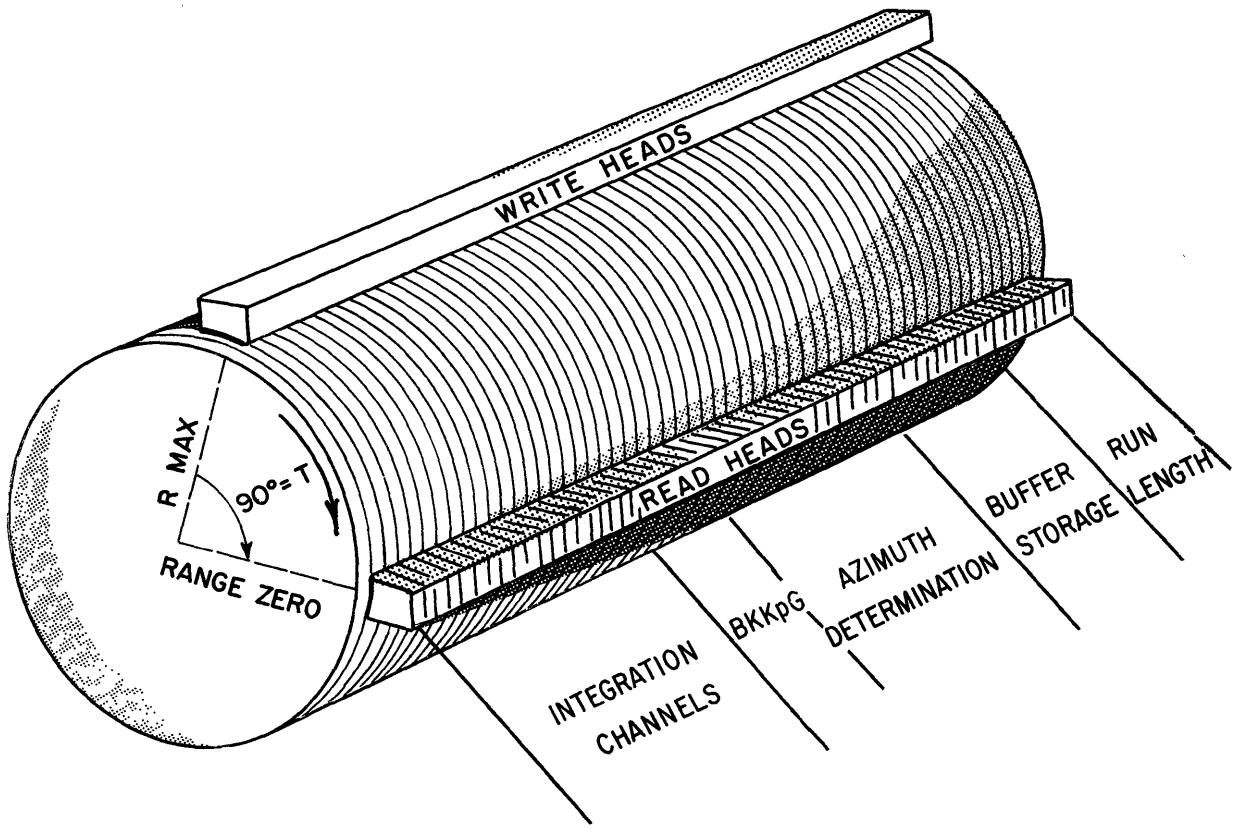


Fig. 8. Drum Memory for Radar Data Processing.

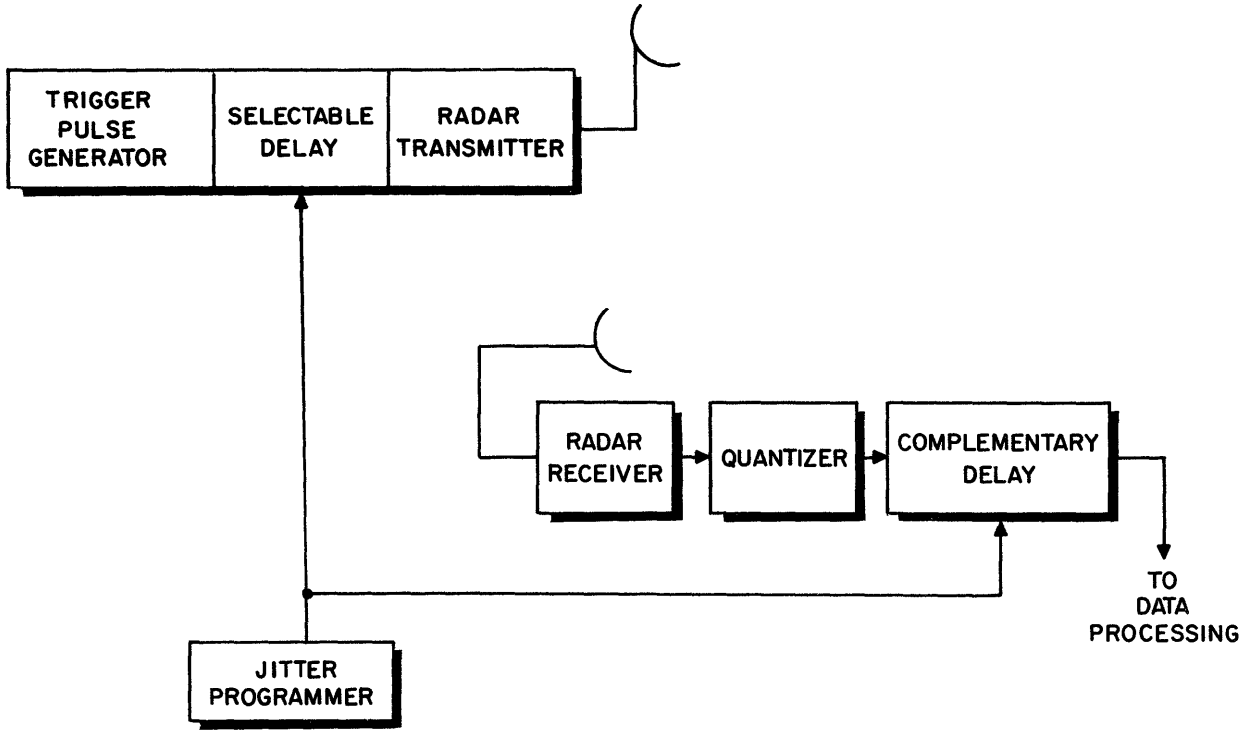


Fig. 9. Digital Jitter-Dejitter System.

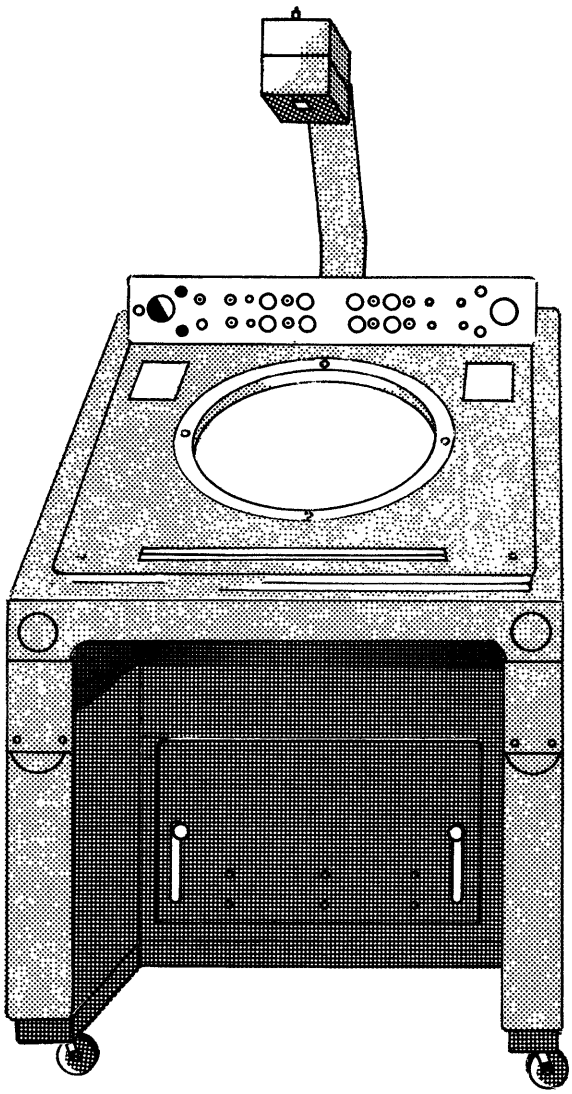


Fig. 10. Manual Clutter Mapper.

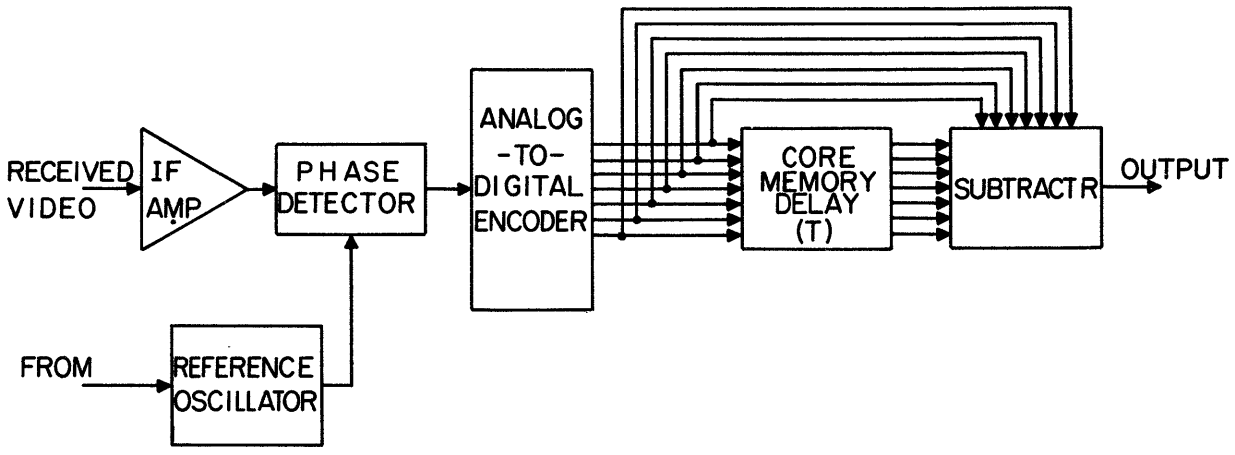


Fig. 11. Digital MTI System.

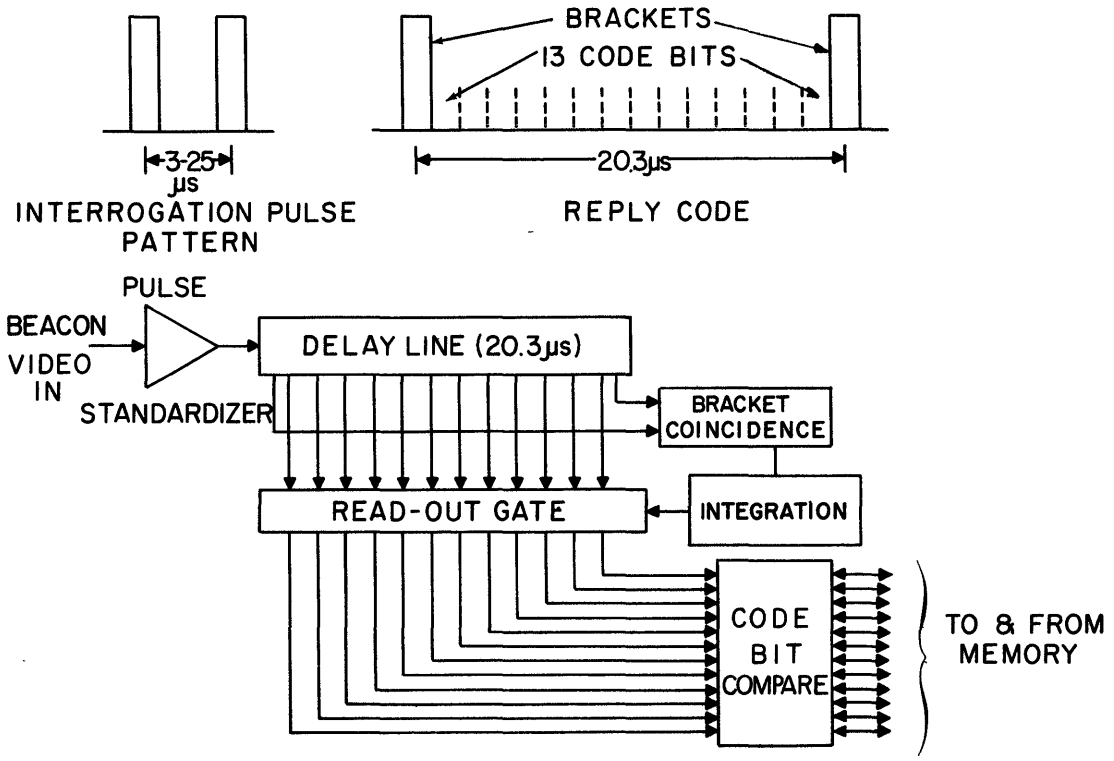


Fig. 12. Elementary Beacon Data Processing System.

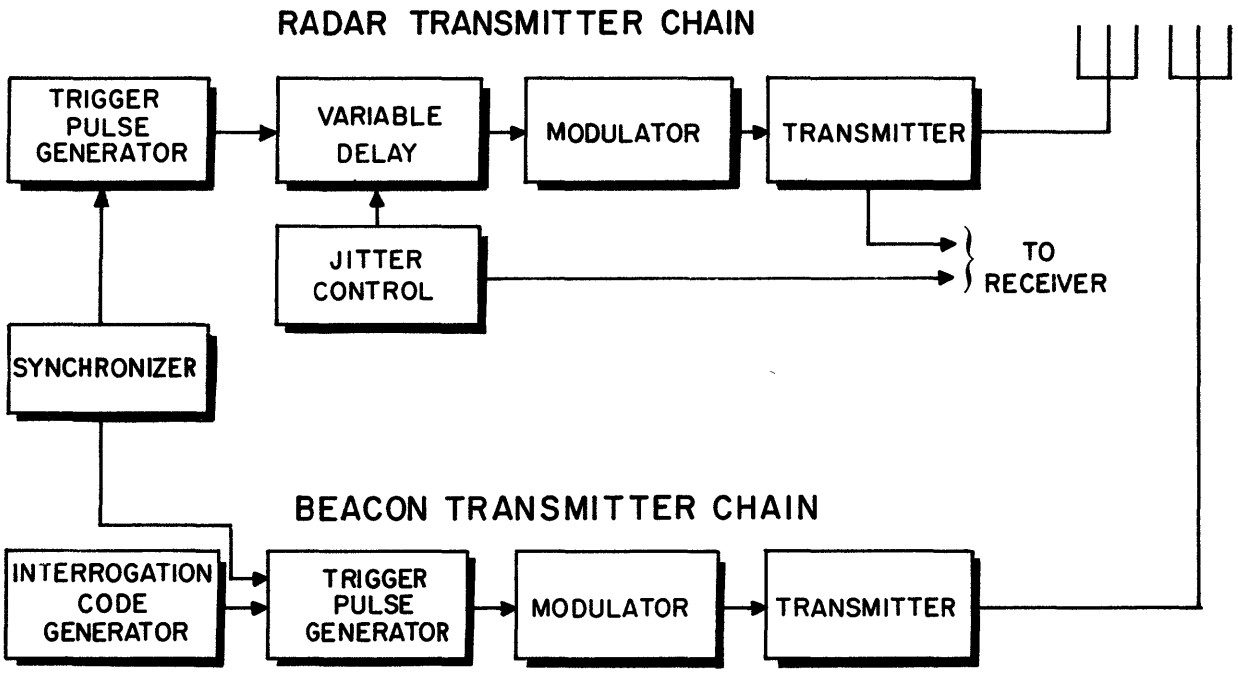


Fig. 13. Radar and Beacon Transmitter Systems.

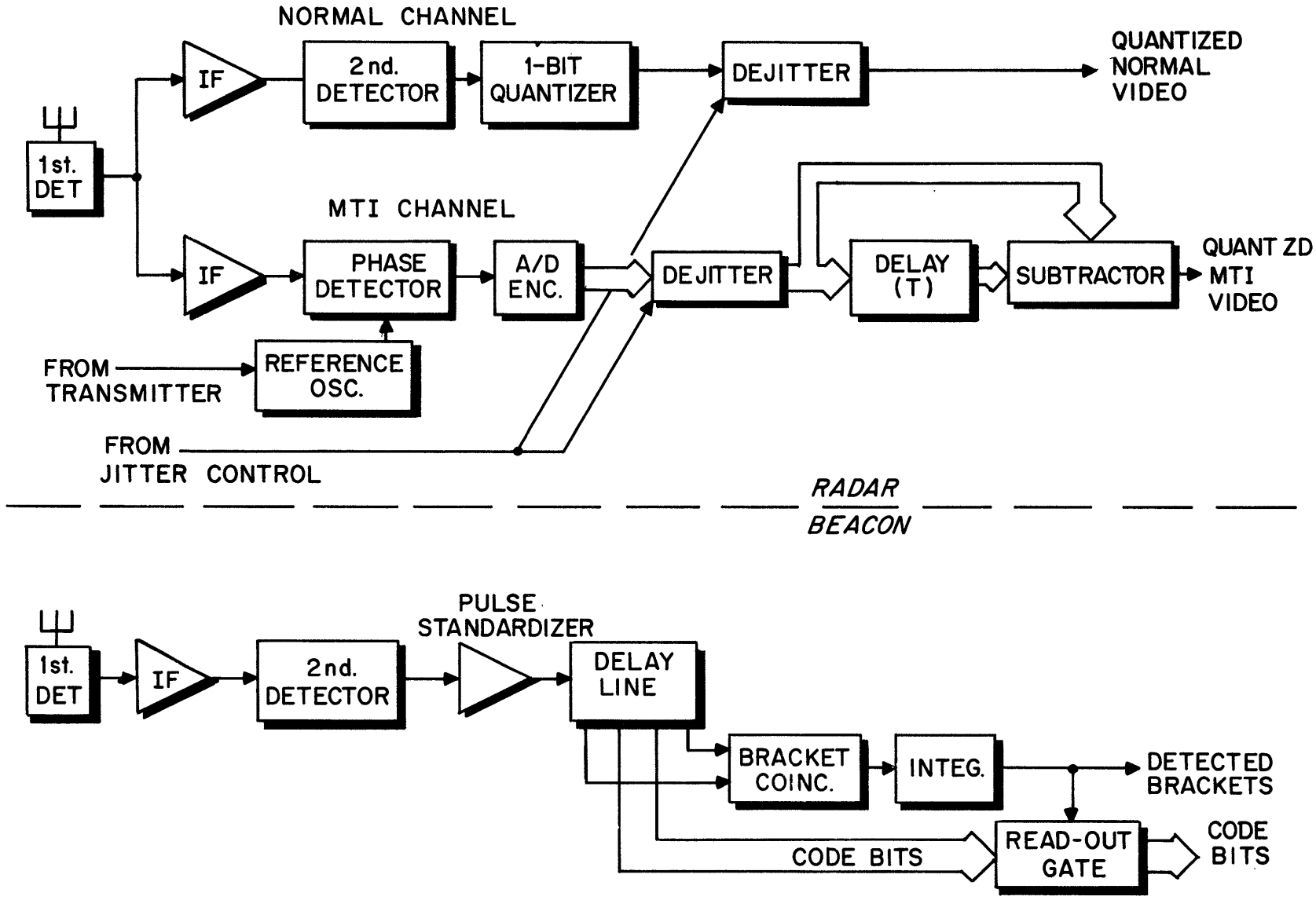


Fig. 14. Preliminary Video Processing.

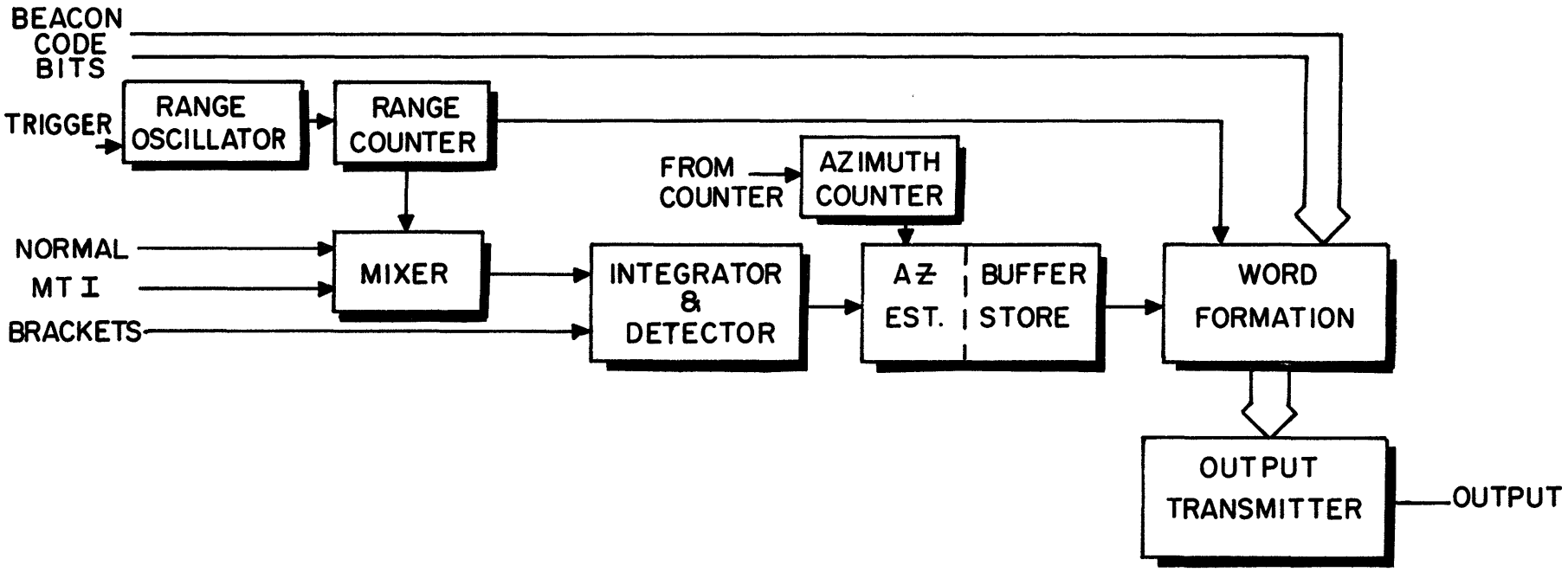


Fig. 15. Final Video Processing.

ORGANIZATION AND PROGRAM OF THE BMEWS CHECKOUT DATA PROCESSOR

A. Eugene Miller
Senior Member, Technical Staff
Auerbach Electronics Corporation

Max Goldman
Manager, Checkout, Monitoring, and Electrical Integration, BMEWS
Radio Corporation of America

Introduction

The Ballistic Missile Early Warning System (BMEWS) Checkout Data Processor (CDP) is probably the first medium-size digital processor to perform the real-time, on-line checkout of an entire operational radar detection and processing system. This paper is the first to describe the unique organization of the BMEWS CDP and the unusual structure of its program. It also states many of the detailed characteristics of the CDP. The Checkout Data Processor has several modes of operation. As a point of reference, the mode which inserts a "realistic" sequence of events into BMEWS is focused upon in the following discussion.

The CDP has two functional memories; one for storing constants and instructions and one for storing data. The means for jointly using these two memories and still maintaining the flexibility associated with single memory machines is brought out. The features tailored in the CDP for efficiently handling its problem are emphasized. They include real-time program interrupt signals and a complex Input-Output System. This Input-Output System, as well as communicating with over a dozen other digital data handling devices, has more than 250 separate addresses.

The structure of the CDP program is the other area focused upon by this paper. The material covered describes the three separate programs which run in an interwoven fashion. This interweaving and the effect of the real-time program interrupts are brought out.

This paper is the first comprehensive public description of this major subsystem of the Ballistic Missile Early Warning System.

Role of the CDP in the BMEWS

The BMEWS Checkout Data Processor has the primary purpose of determining the operability of BMEWS by inserting either test patterns or a "realistic" sequence of events into the system, and then evaluating the BMEWS on the basis of its response.

Figure 1 illustrates the way the CDP fits into the BMEWS. In the normal or real situation, a radar data take-off collects radar returns on a target and assembles these to produce a report. Each report is transferred to the missile impact predictor which stores and correlates the various radar data take-off reports. On the basis of these reports, the missile impact predictor prepares various reports such as a report containing the values of the parameters associated with an observed missile trajectory.

The CDP controls the insertion of a simulated sequence of returns into the front end of BMEWS. At the same time, the test tag is issued to the appropriate radar data take-off to indicate that the return is simulated. The test tag remains with the data throughout the system to differentiate test information from real information. As the radar data take-off correlates the simulated returns, the report that is produced and sent to the missile impact predictor, is also sent to the CDP.

The missile impact predictor uses the data take-off reports to produce reports of its own. These reports are also sent to the CDP.

Organization of the CDP

It is convenient to consider the CDP as being composed of five major subsystems; Wired-Core Memory, Coincident-Current Memory, Arithmetic and Logic Unit, Input-Output System, and Control System. A block diagram of the CDP subsystems and their interrelation is illustrated in figure 2.

Wired-Core Memory

The Wired-Core Memory, which is a form of the Dimond ring translator, stores the program and constants used by the CDP. It contains 4,096 storage locations. A Wired-Core Memory was chosen for two reasons; namely, speed and reliability. Figure 3 illustrates the instruction word format as it appears in Wired-Core Memory. Since the CDP has two diverse types of memories, the locations of each of the types of memory are called by different names. The Wired-Core Memory locations are called "locations" and the Coincident-Current Memory and input-output locations are referred to as "addresses".

Bits X_0, X_1, and X_2 are used to specify the operation modification. Four configurations of X_0, and X_1, and X_2 are interpreted as follows: when X_0 is zero the operation is specified by X_3 through X_7 with X_8 through X_{19} specifying the accompanying address or location. Bit X_2 is the parity bit associated with X_3 to X_7. Three other configurations of X_0, and X_1 and X_2 when X_0 is one specify an operation to be performed with a constant. In these cases X_4 to X_{21} are used to specify the constant. These bits are actually transferred into the arithmetic unit as data.

In the normal operations, i.e., when the operation modifier indicates that X_3 through X_7 is to be interpreted as the operation and not part of a constant, X_{20} through X_{22} are used to indicate which of the index registers are to be used with the instruction. There are three independently used index registers. There is one exception to the above statement in which X_{20} through X_{22} indicates a specific bit in the accumulator which is to be tested.

One of the tailored features of the CDP is concerned with performing table lookups of constants stored in the Wired-Core Memory. The sequence of instructions is initiated by a normal instruction which performs two functions. The first function is to store the location of the next instruction in a fixed address of the Coincident-Current Memory. The second function is to jump to the address specified in the instruction itself. This address may be modified by the various index registers. The jump leads to an instruction which contains an operation modification. This operation modification adds the constant specified in the instruction into the arithmetic unit and then control is transferred to the location specified by the fixed address in Coincident-Current Memory. This same technique can also be used to enter and leave sub-routines as well as perform table lookups.

Another feature added to the instruction repertoire to give desired flexibility in using the Wired-Core Memory is an indirect jump. This instruction, in Wired-Core Memory, specifies an address in the Coincident-Current Memory which contains the location in the Wired-Core Memory to which control should be transferred. Thus, variable linking can be accomplished by changing the contents of the Coincident-Current Memory address involved.

Arithmetic and Logic Unit

The Arithmetic and Logic Unit contains an accumulator and associated registers and control circuits for carrying out addition, multiplication, subtraction, division, shifting, masking, and so forth. This unit obtains input data by directly addressing the Input System or via the Coincident-Current Memory. The Arithmetic and Logic Unit also directly sends information to the Output System. The data derived from the addressable inputs or sent to the addressable outputs are used in essentially the same way as data sent to and from the Coincident-Current Memory. For example, one input address contains the azimuth position of one of the radars. The contents of this address can be read into the accumulator as though it were data from the Coincident-Current Memory.

Coincident-Current Memory

The Coincident-Current Memory of the CDP is composed of 1,024 addresses. This memory has several functions. One of these functions is to act as an input buffer for various real-time asynchronous input sources. There are buffers for data reports from the radar data take-offs and reports from the missile impact predictor as well as information from an input magnetic tape. Also, there is an output buffer which is used to store information required in simulating RF returns. When a data word is required by or available from an outside source, the program is interrupted for a memory cycle during which time this data word is removed from or stored in an allocated address in the Coincident-Current Memory. The interruption occurs without the knowledge of the program.

Other functions of the Coincident-Current Memory include storing intermediate results and control information developed and used within the program. Also, the indirect address feature and standard address feature previously mentioned are facilitated by the use of Coincident-Current Memory.

Input System

The Input System can be considered as composed of two diverse parts. There are demand inputs which supply information such as the radar position vector and console commands, and asynchronous inputs which supply magnetic tape information and system reports. Each separate demand input is associated with a particular address. As was stated previously, these addresses can be read or acted upon in a normal fashion. There are over 180 addressable inputs which fall into 36 different classes of information. The Input System has the facility for communicating with well over a dozen other digital data handling devices including a pair of IBM 7090's.

The asynchronous inputs are stored in the Coincident-Current Memory by interrupting the program control over the memory for single read-write cycles. Beside information from the input magnetic tape, the CDP receives, by means of this Coincident-Current Memory

interrupt feature, 12 different types of messages from other processors in the system with an average of six to seven items per message type.

Output System

The Output System has the same dicotomy as the Input System. There are over 75 addressable outputs which fall into about 15 classes. A major portion of the Output System is associated with a target simulator which actually produces the simulated returns and accompanying test tags. Figure 4 shows one of the target generators of the target simulator. An output address is associated with a digital-to-analog converter (DACON) which is used to control the amplitude of the desired output signal. Another DACON, which is loaded by the program, controls a variable frequency oscillator (VFO). The modulator controls the amplitude of the output signals generated by the VFO. The range value is placed into a counter by the program at some time prior to the initiation of the radar main bang[1]. At the beginning (leading edge) of the main bang, an oscillator is connected to this counter providing an output at an appropriate time to simulate a return at the range desired. This counter output lasts for the duration of the desired return. As a result, the output of the target generator is a pulse of the correct amplitude and frequency at the desired time to simulate the return from a target at the corresponding range. It is worth noting at this point that the CDP acts as the controlling element in a feedback loop for the variable frequency oscillator. The output of the VFO is fed into a counter for a fixed period of time. This average frequency is sampled by the program to produce a new value to obtain the desired frequency. The new value is placed into the digital to analog converter which controls the VFO. This subject will be discussed in a later paragraph.

A summarizing fact which further indicates the unusual complexity of the Input-Output System is that there are more than 1,300 connections for the addressable inputs and outputs alone.

Control Unit

The Control Unit, besides containing the facilities usually associated with control functions of internally stored program digital computers, has five features used in interesting ways. The first of these features, which was mentioned previously, is the ability to recognize signals from the Input-Output System. These signals indicate to the Control Unit that a memory cycle is to be usurped for asynchronous input-output reasons. The Control Unit has a built-in priority system to handle simultaneous input-output requests as well as the address control for storing the data.

Another feature, also previously mentioned, is the incorporation of a timer in the Control Unit to establish the sampling period for measuring the variable frequency oscillator outputs. This is a two-phase timer that allows the sampling of the VFO output to take place for a fixed period of time followed by a period of time during which no sampling takes place. This dead period is used by the program to compute the necessary corrections and also to apply new inputs to the VFO's.

One of the more highly tailored features of the CDP is the three index registers and their use. The discussion of these index registers is included in the Detailed Characteristic Section. Other features associated with the Control Unit are the inclusion of a special tag register and the presence of program interrupt signals which relate to the occurrance of the radar main bang. These features are discussed in the Programming Section.

Detailed Characteristics

A binary numbering system is used in the BMEWS CDP with a data word length of 18 bits plus one bit for parity. Accessing an instruction from the Wired-Core Memory and modifying the address by index registers requires four microseconds. There are two memory accesses per instruction: one memory access of the Wired-Core Memory for the instruction itself and another access from the Coincident-Current Memory or input-output address. In either case, 8.8 microseconds is involved in the access and manipulation of data for basic instructions. Therefore, an instruction such as an addition requires 12.8 microseconds to complete; including the instruction access and modification of the address by index registers.

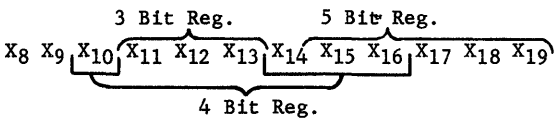
The CDP uses 27 out of the 32 possible instructions facilitated by the five bits of the operation code. There are also the three operation modifications for use with constants.

The Wired-Core Memory contains 4,096 - 23-bit words. The Coincident-Current Memory contains 1,024 - 19-bit words. Both memories use a binary addressing system and all data and instructions are addressable by words.

The CDP contains three index registers - a three-bit register, a four-bit register and a five-bit register. The contents of each of these registers are logically added to the address contained within the instruction.

1. Reference to a radar main bang is confined to the leading edge. Reference to an interpulse period is defined as the time between successive main bangs.

There is no time penalty associated with the use of the index registers. The index registers mix into an address in the following fashion:



There are specific operations which use indirect addressing only. There is no time penalty associated with such operations, that is, they require 12.8 microseconds to carry out.

The BMEWS CDP is capable of simultaneously carrying out input operations while performing arithmetic computations. For example, the program performs a single instruction which is a start-tape command. The program then ignores the tape input operation, while performing arithmetic operations, until an appropriate time when an inspection of the tape input buffer reveals the required information.

Information is stored on the input magnetic tape as 6-bit characters plus a parity bit (character parity is odd). There are 120 characters to a tape record. Three 6-bit characters are assembled by the Input System to form one 18-bit CDP word. Thus, the input magnetic tape buffer size (in Coincident-Current Memory) is 40-CDP words. The Input System supplies an entire word to Coincident-Current Memory during each transfer operation. The nominal tape reading speed is 15,000 characters per second.

The output console printer is controlled by a 40-bit buffer which is addressable by the CDP. These 40-bits are composed of ten 4-bit characters. The printer operates at twenty 10-character lines per second.

The clock rate of the CDP is 1.25 megacycles. It contains about 8,000 transistors and between 40,000 and 50,000 diodes. The CDP is mechanized by NOR type transistor-diode logic.

Structure of the CDP Program

The CDP program must carry out the functions associated with several checkout modes. Each of the modes is designed to evaluate the BMEWS from a slightly different point-of-view. Some modes insert "realistic" raids into the BMEWS while other modes insert test patterns into the BMEWS. The CDP program is composed of three parts; the simulation program, the evaluation program, and the executive program. These programs are used in all of the checkout modes.

Each of the programs is composed of routines. The mode determines which routines of each program are to be used for the processing.

Figure 5 illustrates the organization of the CDP program. The processing proceeds from the executive program to the simulation program and back to the executive program. From the executive program the processing then proceeds to the evaluation program. The solid lines are used to indicate a programmed connection between separate programs. The dotted lines connecting the evaluation program to the executive program signifies an automatic transfer caused by the program interrupt feature.

The program interrupt feature is best explained in relation to figure 6 which illustrates the CDP program timing. At each radar main bang the CDP program is interrupted and control is transferred to the executive program. This interrupt should only occur during the evaluation program. A special tag register, previously mentioned under the Control Unit, carries an identification of the program which is in process. The executive program after the transfer of control has taken place, caused by the program interrupt, inspects the contents of the tag register to insure that the interrupt occurred during the evaluation program. The executive program stores the interrupted state of the CDP when the interrupt occurred, and then interprets the mode and sets up various linkages required for the mode if a mode change has occurred. Control is then transferred to the appropriate simulation routine.

When the scheduled simulation routines have been completed, control is transferred in a programmed fashion back to the executive program. The executive program determines where in the evaluation program to resume processing and sets up the state of the CDP for the resumption. The evaluation program is then resumed. As will be seen shortly, the evaluation program is essentially a non-ending program so that it continues processing until the next program interrupt signal.

The executive program coordinates the information flow between the simulation and evaluation programs. The executive program accomplishes this task by having its routines perform such functions as determining the mode and inspecting to see that the program interrupt signals are in the proper sequence, connecting the proper simulation and evaluation routines used in a given mode, and determining the malfunction of the CDP by inspecting the contents of error registers, etc.

The simulation program is composed of routines which generate information defining the signals to be injected into the system. This information is used to control the checkout target simulator which actually creates the signals and sends them into the front end of the BMEWS. The routines of the

simulation program are of such nature that they must be completed by specified times in order to be of use. Therefore, these routines are synchronized by the program interrupt signals.

The simulation program performs three major functions. These are:

(1) Determines the system status (connections between BMEWS major subsystems),

(2) Controls the target simulator,

(3) Prepares the values for the parameters of anticipated messages for use by the evaluation program.

The evaluation program processes the information received by the CDP from the BMEWS in order to evaluate the operability of BMEWS. There are also evaluation routines used to organize information to be printed out; this information is pertinent to the evaluation of the operability of BMEWS. Likewise, there is a routine in the evaluation program for determining CDP failures. Evaluation routines, unlike the simulation routines, do not have to be kept in step with events of each main bang period, but must only meet time requirements in the large. The evaluation program, once started, continues from evaluation routine to evaluation routine until interrupted by a program interrupt signal.

The evaluation program is divided into three priority classes of routines; routines corresponding to a given mode of operation of the first class precede those of the second class which in turn precede those of the third class within a main bang period. The highest priority classes are always initiated in each main bang period (or continued if it has been interrupted, see below). This highest priority class is initiated by the executive program after program interrupt 1 (see figure 6). The three sets of evaluation routines are called Class I, Class II, and Class III with the highest priority being Class I, the next priority Class II, and the lowest priority Class III. Thus, the uninterrupted processing sequence would have the Class I routines processing all their applicable data, followed by the Class II routines processing all their applicable data followed by the Class III routine. The Class III routine (error detection routine) is a non-ending routine, i.e., the "end" of the routine leads back to the beginning.

The CDP program is organized to begin the processing of a given class of evaluation routines at the point of interruption of the interrupted routine of that class. If a given evaluation process was not interrupted then processing begins at the first routine of that class.

Figure 5 illustrates the organization of the three classes of the evaluation routines in the CDP program. The possible returns to the evaluation program are shown inside the executive program. Note that there is a path from each interrupt to the starting of the Class I routines each main bang.

The evaluation program performs three major functions. They are:

(1) Process messages received from the rest of BMEWS - using anticipated values from the simulation program.

(2) On the basis of (1) - turn on appropriate console lamps and printout appropriate information.

(3) Check the operation of the CDP itself.

The evaluation of system reports is illustrated in figure 7. The evaluation routine picks up the value for each of the parameters in the received report and subtracts the corresponding anticipated value. A new message is composed which contains the message type, the target tag, the anticipated value for each of the parameters and the calculated difference between the received value and this anticipated value. This new message is a candidate for printout. The criterion for printing the message is that the deviation of the value for at least one of the parameters is larger than the specified tolerance. If all of the deviations are within the specified bounds, normally, the message is discarded. However, subject to a switch setting on the console, all such printout candidates can be printed-out for the purpose of data collection. An out of tolerance deviation is marked appropriately to identify this situation to the operator.

Summary

A program interrupt occurs at the beginning of the radar main bang (figure 8). This interrupt causes the executive program to store the state of the CDP and proceed into the simulation program. The simulation routines, which are carried out at this time, control the variable frequency oscillators of the target simulator and check to see that the BMEWS status has remained stable. That is, there has been no change in the way the major subsystems of BMEWS have been linked together. The completion of these two routines returns the program to the evaluation program by way of the executive program.

The evaluation program inspects the Coincident-Current Memory for information coming into the area reserved as system report input buffers and, if information has

been received, compares these reports with anticipated reports also stored in the Coincident-Current Memory by the simulation program. The evaluation program is eventually interrupted by program interrupt 1. Console switches are inspected to see that the CDP is still to remain in the present mode or change modes. The simulation program is then initiated.

The simulation routine which is carried out at this time calculates the inputs required by the target simulator at the end of this main bang to simulate returns following the subsequent main bang. This calculation is based upon the system status, the particular position of the various beams, and target data which had previously been stored in the Coincident-Current Memory by the magnetic tape portion of the Input System. The simulation routine, prior to starting its calculation, initiates the reading of a record from the input magnetic tape for use during the next inter-pulse period. Data used in this inter-pulse period had been read in during the previous inter-pulse period. The results of the calculation of this simulation routine are stored in Coincident-Current Memory ready for transfer to the target simulator at an appropriate time.

The evaluation program is then re-entered via the executive program. Here the executive program restarts the Class I routines if they had not just been interrupted. If they had just been interrupted by program interrupt 1, the interrupted situation is resumed. Again the evaluation program continues until the next program interrupt occurs. At this point, the executive program leads to the simulation routine that transfers the information previously calculated for the target simulator into the target simulator.

The target simulator must be quiescent during the period of time that the program transfers new parameter values into the range counter. Therefore, the maximum range that can be simulated is limited by the period of time associated with this transfer. In actuality, program interrupt 2 marks the end of time when the target simulator is capable of producing a simulated return during the main bang.

Thus, each main bang, data taken from the input magnetic tape during the previous main bang and stored in Coincident-Current Memory is used together with system information to produce control information for the target simulator. This information is transferred into the target simulator at the end of the main bang. During the next main bang, while this same calculation process is being repeated, the target simulator inserts into the system, if required, simulated RF returns and corresponding test tags. This process continues main bang interval after main bang interval.

The radar data take off correlates the inserted signals and issues reports to the missile impact predictor. These same reports are shipped to the CDP and stored in the appropriate area of the Coincident-Current Memory. The evaluation program continues to inspect this portion of the Coincident-Current Memory sensing for such reports. When a report is received, it is evaluated by using the anticipated report produced by the simulation program. If an out of tolerance situation is discovered, information is shipped out to the operator by means of the printer.

Radar data take off reports are collected by the missile impact predictor and used to produce various other reports. These reports are shipped to the CDP and stored in the Coincident-Current Memory. In a similar fashion, the Checkout Data Processor compares these reports with anticipated reports and indicates to the operator the result of such processing. In this way, the Checkout Data Processor using its equipment and its program in an interwoven fashion, generates simulated RF returns, injects these returns into the front end of BMEWS, receives the effect that these signals have on BMEWS, and on the basis of these effects, evaluates the operability of the overall system. Finally, the checkout device, being a major part of BMEWS, evaluates its own operability.

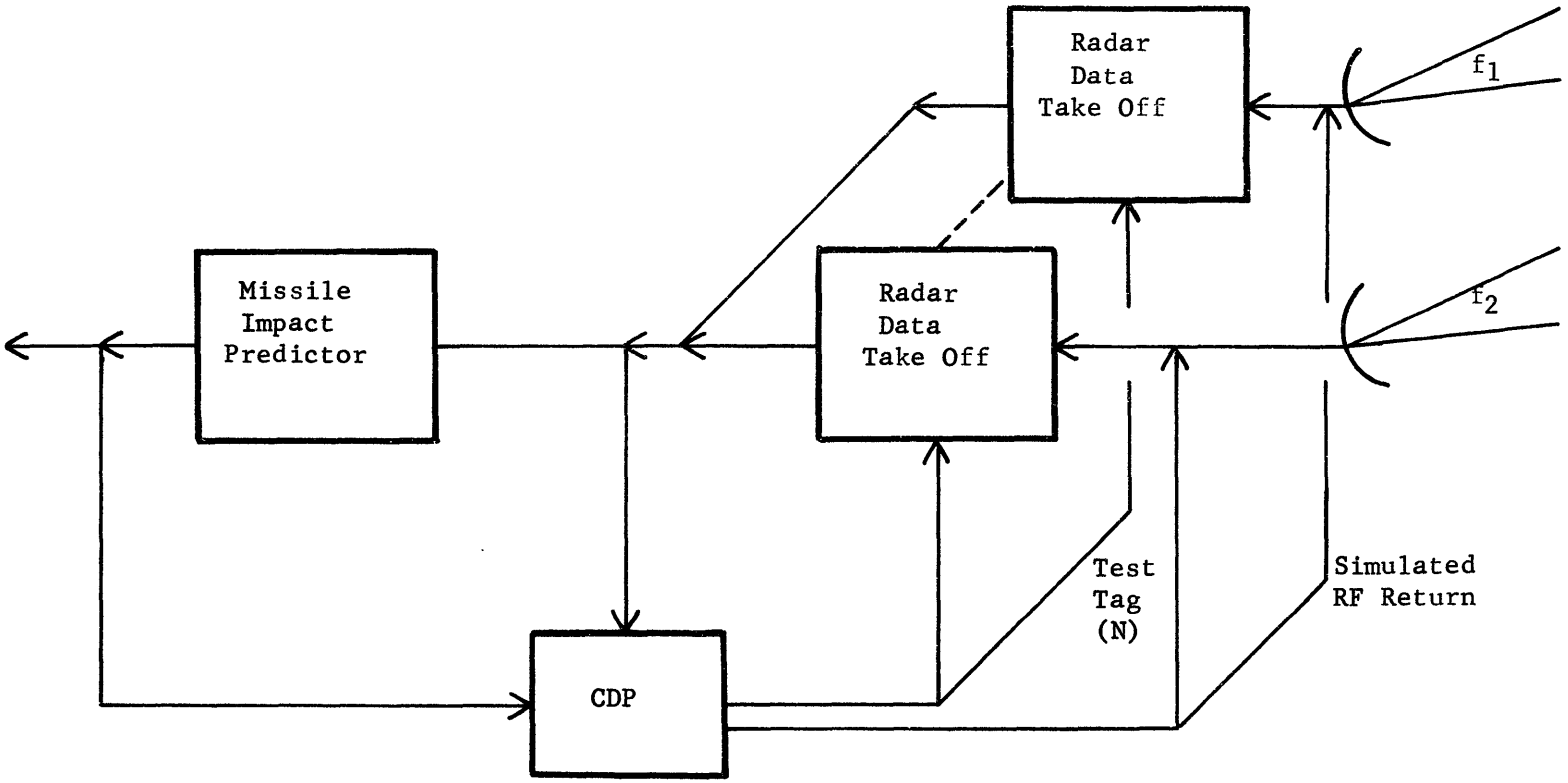


Figure 1. CDP in the BMEWS

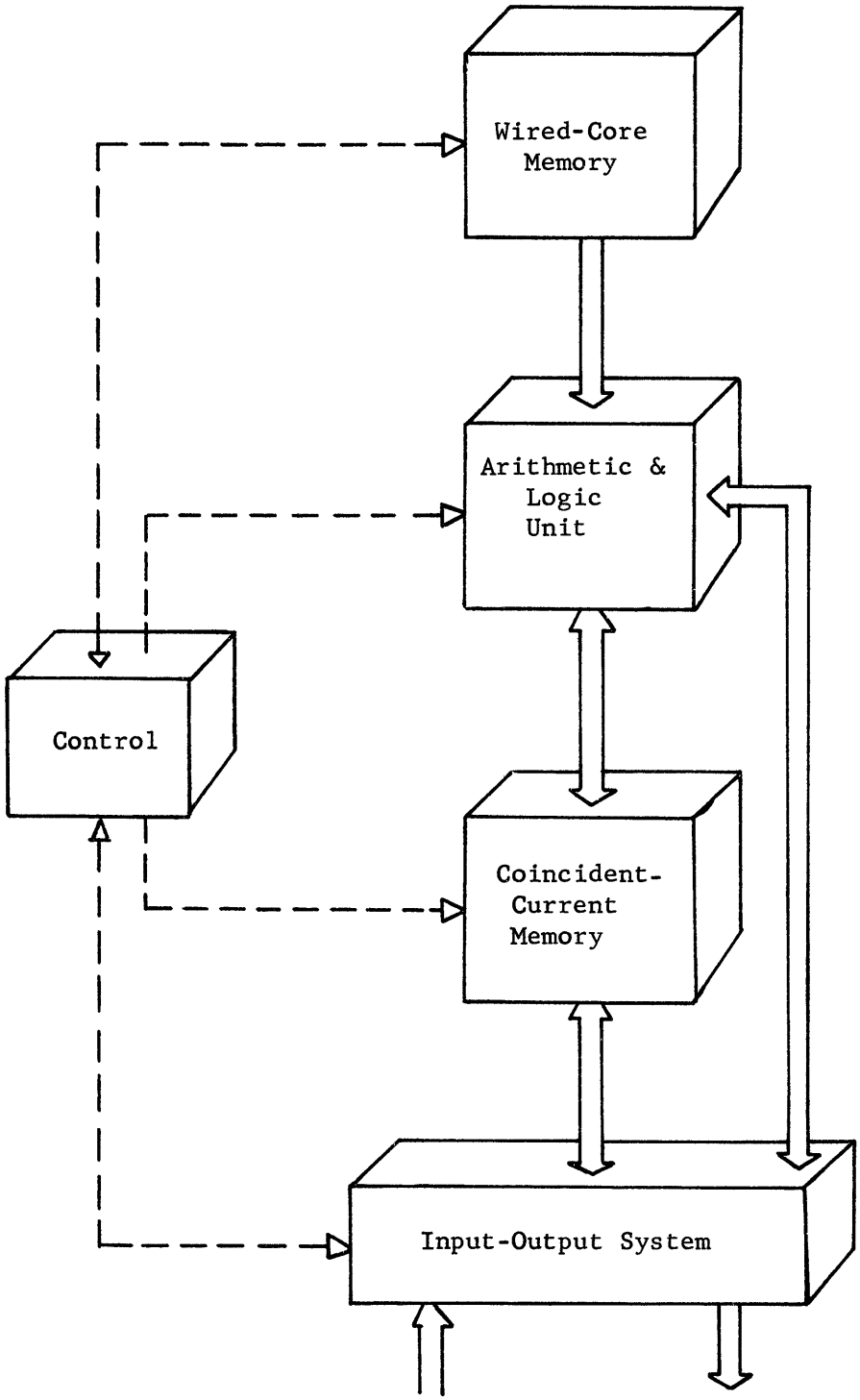


Figure 2. Organization of the CDP

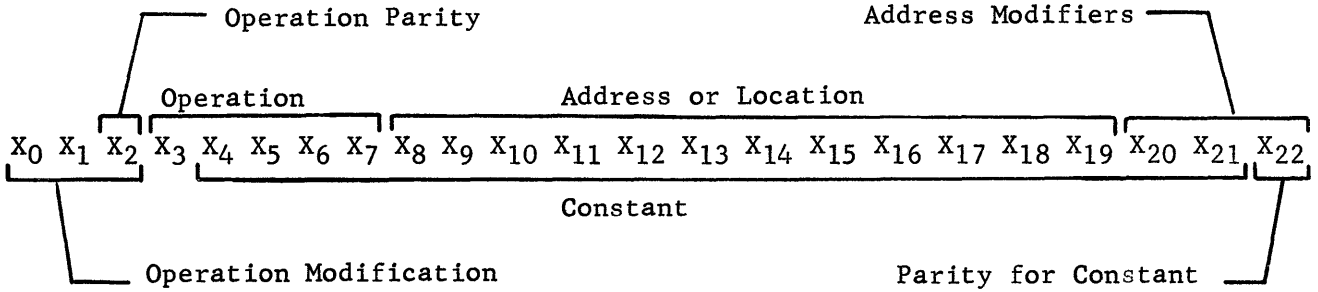


Figure 3. Wired Core Memory Word Format

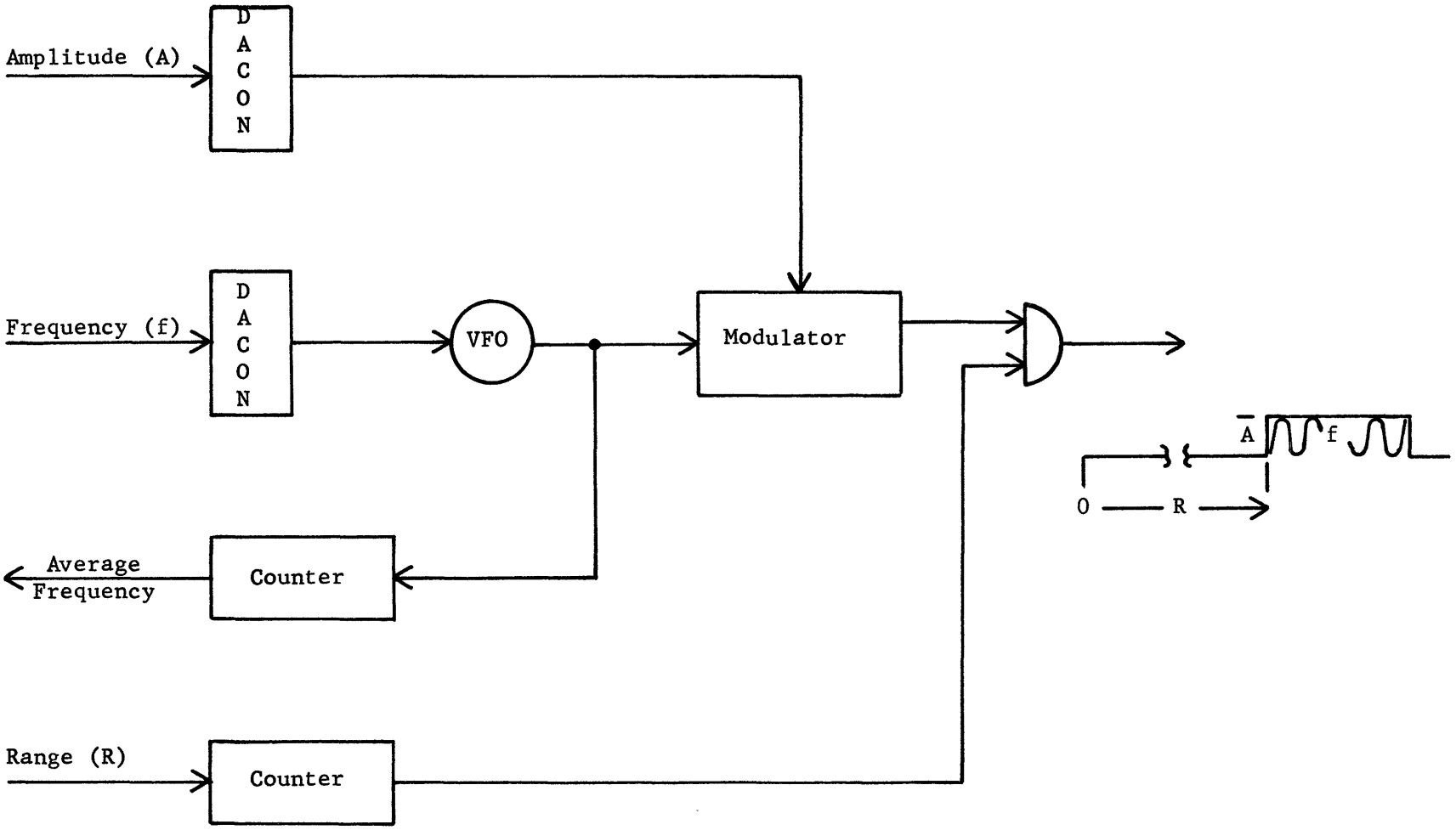


Figure 4. Target Generator

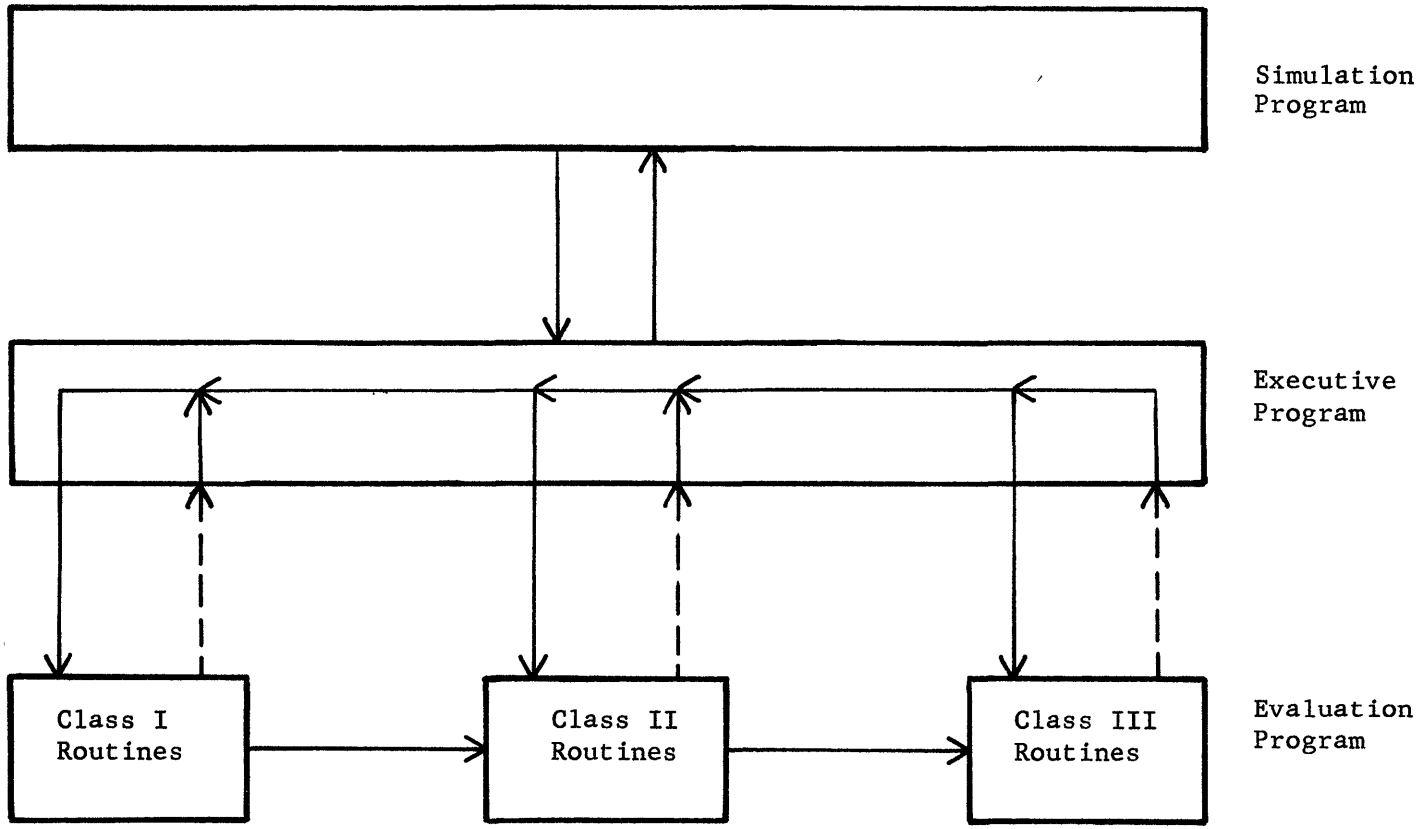


Figure 5. Organization of the CDP Program

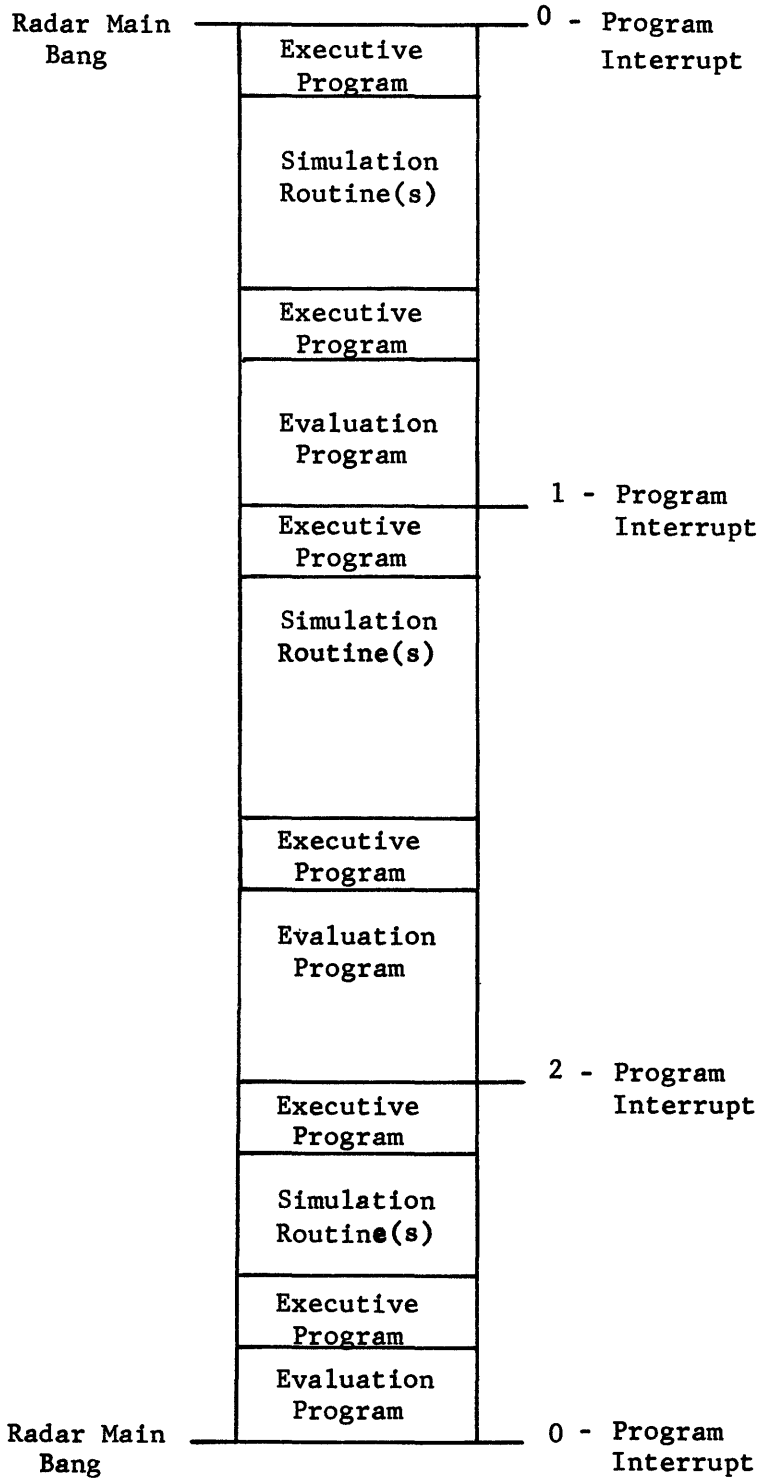


Figure 6. CDP Program Timing

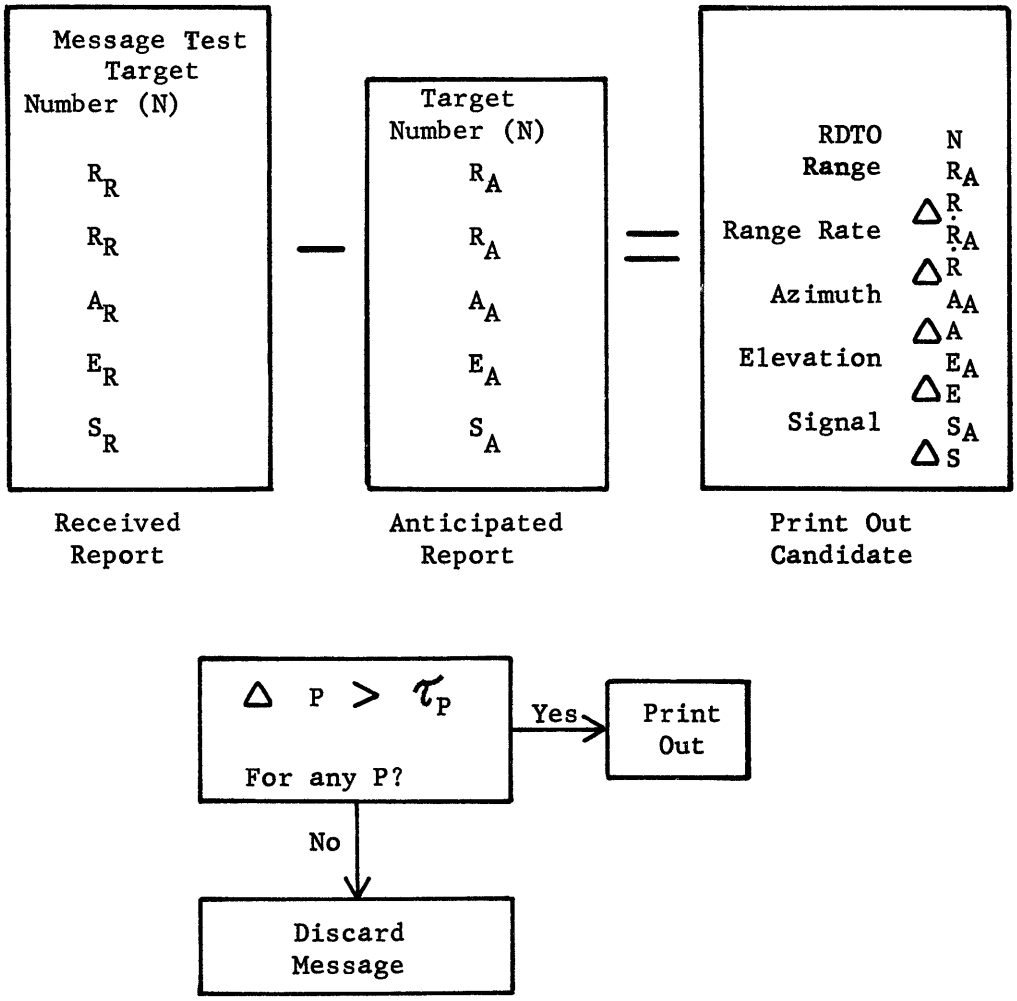


Figure 7. Evaluation Process

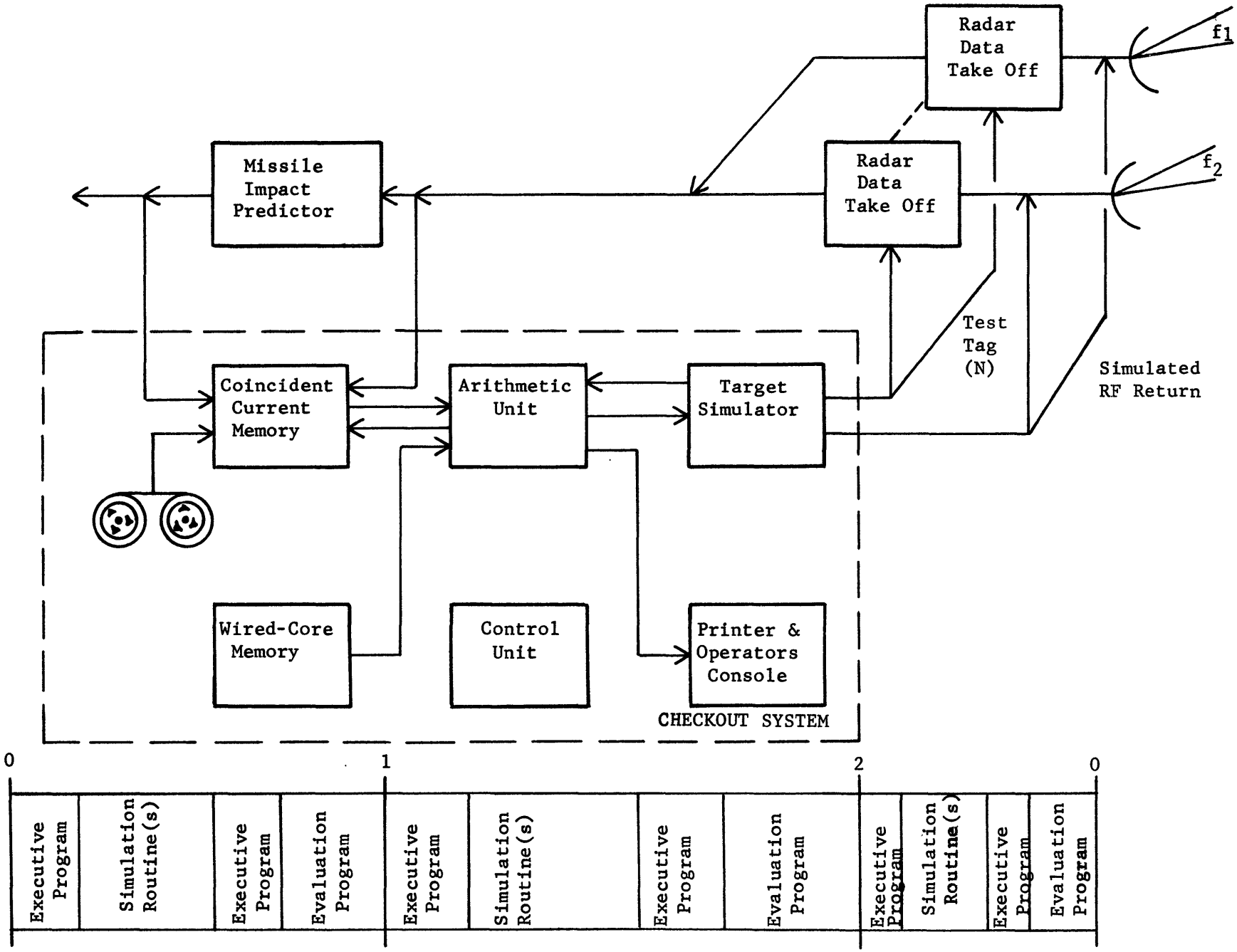


Figure 8. Summary

HIGH-SPEED DATA TRANSMISSION SYSTEMS

R. G. Matteson
Stromberg-Carlson
a Division of General Dynamics Corporation
Rochester, New York

Introduction

For many years data and messages have been transmitted from point to point over slow speed data communication systems by telegraph. There is a rapidly growing increase in the use of centralized data processing equipment in large corporations however, making increasing demands upon existing data communication systems. Development has been carried out at Stromberg-Carlson during the past few years toward increasing the capabilities of the standard telephone facility for the transmission of data at higher speeds and with greater reliability. It is felt that this equipment will have widespread use in the business data field, the scientific field, the automatic control field, and for military data communication systems. Typical components required for the transmission of data over telephone lines at high speed include input/output equipment, buffer converters, modulator/demodulators, and of course a transmission path. Stromberg-Carlson has developed a modulator/demodulator unit using a unique modulation principle specifically designed for minimizing the effects of the various types of distortion and interference associated with wireline systems. In addition, an installation has been completed for the Convair Division of General Dynamics Corporation which transmits data in the form of punched card information over a telephone line more than 200 miles long, and records the information on magnetic tape in a format compatable with IBM 704 programming.

Applications

Since the processing of data by digital computers is becoming common practice throughout many types of endeavors, the applications for the transmission of data at high speed over standard telephone facilities appears to be very wide spread. Some of these areas will be discussed in the following paragraphs.

Many corporations are facing a decision today between large computers at a central location in the corporation or many small computers geographically separated at the various divisions. If a centralized computer facility is required for performing all the various operations and data processing for a corporation, the data must be transmitted from remote divisions of the corporation into the central data processing center. For a moderate size corporation this data can reach significant proportions and can only be transmitted by air mail or many slow speed data transmission systems operating in parallel at the present time. The availability of equipment to transmit data at higher speeds over telephone facilities which are normally available for telephone communications will assist in solving the data handling problem. For the corporation which decides to use individual smaller computers at each division, data transmission can still be of considerable benefit. For these corporations, high speed transmission of data can mean faster reporting between divisions and the possibility of using remote computers if the local computer becomes overloaded during peak operations.

Many corporations are looking towards automatic data collection, data acquisition and transaction recording systems to accelerate the over-all data processing and reporting cycle. These systems will be used for such things as the compiling and accumulation of data concerning job moves, stockroom transactions, job changes, inspection results and attendance recording. This data may be recorded for several manual input units at an intermediate collection point. The data must then be transmitted from this intermediate collection point to a central collection agency

(Figure 1). For a sizeable corporation, the amount of data transmitted into the central recording facility will be considerable. Transmission must therefore occupy a minimum of time in order to be able to submit all data from each of the intermediate collection facilities, requiring high-speed transmission of data over standard wireline facilities.

Some types of businesses such as banks and insurance companies have requirements for large data retrieval systems. In this case, a central file of information, upon receiving an inquiry from a remote station, will transmit the data requested in the inquiry. In this example, the amount of data transmitted during the inquiry is small, but the amount of data transmitted during the reply by the data file may be large indeed depending upon the particular application. In order to satisfy a number of independent inquiries in a reasonable amount of time, data will have to be transmitted at high speed over wireline facilities.

For the solving of scientific problems, large scale computer facilities are required in order to handle the more complicated problems expected. For some of the large computer facilities required, it is economically impossible to duplicate a facility at remote corporation installations. In this case, remote facilities can use a large scale computer facility for the solution of scientific problems by transmitting the program and input data by high-speed data transmission systems to the central computer facility (Figure 2). The problem solution can then be transmitted back to the originating site.

It is also occasionally possible to break down a large problem into subproblems which can be worked independently by separate computer facilities. In this case, various computers throughout a wide-spread corporation can be used for solving parts of the same problem by means of high-speed transmission of data over telephone facilities.

The solutions to scientific problems transmitted over the high-speed data transmission system can be tabulated, reported or plotted by the use of direct on-line high-speed printing equipment.

For applications where digital computers are used in the automatic control of process or operating conditions, much data is transmitted between the operating system and the computer control system. In the case of process control computers, the requirement is for the transmission of measured values of the operating system, the transmission of control signals to change the operation of the system and the transmission of data to various indicators allowing manual supervision of the operation of the system.

Operation of organizations such as gas and pipe lines, and railroads requires the transmission of operating data over long distances. This data can be economically transmitted at high-speed over the same wire-line facilities normally used for voice communication.

Many requirements exist for the transmission of considerable amounts of data for military systems. Military systems such as the SAGE (semi-automatic ground environment) system for the detection, tracking and interception of enemy aircraft requires data to be transmitted between acquisition sites, control centers, and intercepter centers.

With the military striving for higher speeds under tactical conditions, automatic displays and even the computer analysis of tactical situations provides a requirement for the transmission of digital data.

Logistic Systems such as the Air Force COMLOGNET System will be used to transmit a tremendous amount of data concerning logistic information throughout the Air Force.

Equipment Requirements

Any generalized Data Communication System can be broken down into input/output components, buffer converter components, modulator/demodulator components, and the transmission path (Figure 3).

The input/output equipment may consist of punched tape, punched card, or magnetic tape readers, manual input

keyboards, FLEXOWRITER, or electric typewriter equipment. The input/output equipment can also consist of the buffer storage portions of general purpose computer installations.

The buffer converter unit at the transmitting terminal of a data transmission system must accept the data from an input device, and transform this data into the proper format for application to the modulator unit. This may require parallel to serial conversion, temporary storage, and level changing. The buffer converter may also change the language of the data as in the case of card to tape transmission. Also, it may be required to add checking information to the transmitted signals. At the receiving terminal of the Data Transmission System the buffer converter must accept the data from the demodulation unit and convert it into the proper format for recording on the output device used. This operation may mean serial to parallel conversion, temporary storage, and the generation of additional format information such as inter-record and inter-file gaps normally required for preparing magnetic tape for IBM computers.

Error circuitry is required at the receiving terminal improving or at least indicating the reliability of data transmission. Errors can be detected by checking vertical and horizontal parity bits in the transmitted data, and indicating these to the operator at the receiving terminal. This gives a measure of transmission reliability but does nothing to correct the matter. In the event of an error circuitry can also be included to cause automatic retransmission of the previous block of data. Using this technique, the final received and recorded data will be more reliable by several orders of magnitude than if automatic retransmission were not used. Another technique can be used which would correct certain types of errors in the transmitted data at the receiving terminal without requiring retransmission. By storing a complete block of data and checking horizontal and vertical parity signals, single bit errors in the transmitted message can be corrected automatically. Alternatively, if the information is destined for computer data processing, a program can be incorporated into the data processing routine to perform the same function, thereby simplifying the data transmission system.

The telephone line which will transmit voice information satisfactorily will not necessarily transmit data reliably; for example, impulse noise of very short duration may cause bits of information to be changed, added or deleted in the data being transmitted which could have serious consequences in the business data applications. Similarly, a frequency translation will have serious effects on some types of data modulation techniques, but will hardly be apparent during voice transmission. Certain minimum requirements for line characteristics must be satisfied depending on the type of modulation/demodulation equipment used in the system.

Equipment Description

Data transmission equipment which has been developed at Stromberg-Carlson will be described as examples of the components mentioned in the preceeding sections.

A tape transmission terminal suitable for tape to card, card to tape, and tape to tape systems is shown in Figure 4. A tape transport has been selected for low cost, reliable operation. The buffer converter is designed to accept data from the tape transport, convert it to a serial form, and provide control and synchronization signals to the receiving terminal. These and other functions of the buffer converter are shown in Figure 5. The end of the file is automatically recognized by the buffer converter and the tape transport is turned off. In the event of an error recognized at the receiving terminal, the retransmission signal is recognized by the buffer converter and the data record in error will be retransmitted by reversing the tape transport and transmitting that record over again.

The receiving portion of the buffer converter converts serial input data to parallel data for recording on the receiver tape transport. The data is synchronized to the incoming data by resetting the character bit counter with a start of record character. End of record gaps occurring on the transmitted tape will also be placed on the receiving tape. At the end of a file of data, the receiving tape transport will be stopped.

A modulator/demodulator model SC-301 is used to modulate the serial

train of data in a form for reliable transmission on telephone facilities at 2400 bits per second. The SC-301 converts the input binary information to a trinary form which then amplitude-modulates a subcarrier signal. Advantages of using a trinary, rather than binary baseband signal include the following:[1,2]

(a) Elimination of low frequency components, reducing noise bandwith and easing requirements for transmission circuit characteristics.

(b) Constant average power level in transmitted signal.

(c) Permits use of bistable detector, rejecting noise impulses of one polarity.

At the receiving terminal, the signal is demodulated and regenerated to form the original binary information. A free-running multivibrator is synchronized to the incoming data to provide a clock signal at the synchronous rate of data transmission. A photograph of the SC-301 in a separate cabinet is shown in Figure 6.

A telephone handset is supplied on the front panel of the data communication terminal for intercommunication capability. This enables the operators to coordinate the transmission of data at the beginning and the end of the transmission. A self-test capability allows the operators to check out the transmission link before the start of transmission. Amplitude and time delay equalizers are also provided with the terminal as required to compensate for characteristics of the telephone facility.

A card data transmission terminal is shown in Figure 7. This unit can be used to transmit data between itself and another card terminal, a tape terminal, or a computer input terminal. This terminal has been designed to read punched cards at the rate of 100 cards per minute. The terminal is similar to the tape transmission terminal described above except for the addition of a card buffer module. The card buffer will store all of the data on one card as received on a row by row basis. The buffer will then feed the data character by character into the buffer converter, which then performs functions similar to that performed by the tape terminal buffer converter.

The third type of data transmission terminal developed at Stromberg-Carlson is for transmitting data from computer to computer. This type terminal is shown in Figure 8 and is exactly the same as the card transmission terminal except that the card reader and the card buffer are not required. The buffer converter accepts data from the buffer storage unit of a computer exactly as it would from the card buffer unit. At the receiving terminal, the data is provided to the buffer storage unit of the receiving computer.

Since the tape, card, and computer data transmission terminals all have standard outputs, they can be used interchangeably with each other to form tape to tape, card to card, card to tape, computer to computer, tape to computer, etc. systems. A card to tape system has been installed at the Convair Division of General Dynamics Corporation to transmit data over a 200 mile telephone line between Pomona, California and San Diego, California. This system enables Convair personnel in Pomona to utilize an IBM 704 Computer facility located in San Diego.[3,4] In addition, an SC-3000 high speed communications printer can be used directly on line at the receiving terminal to print out data.

A new type of tape transport is being developed for use in specific applications for data transmission and data collection systems. This tape transport will operate in two modes, a stepping, asynchronous mode or a continuous, synchronous mode. In the stepping mode, the transport can be used with a FLEXOWRITER, manual keyboard, slow speed punched card device or other equipment operating asynchronously. In this mode the unit can be used to record or reproduce data character by character for data recording, collection, and acquisition applications. In the continuous mode, the transport can be used as a substitute for the tape transport discussed previously in connection with the tape transmission terminal.

The transport can therefore be used to store data asynchronously and accumulate it over a period of time. The unit can then be rewound and used to supply the data at high speeds into a data transmission system. In this application a telephone facility can be used for voice communications most of the time, since data can be transmitted at high speed during a small portion of

the day. A block diagram of the tape transport is shown in Figure 9.

Conclusion

The applications for data communication systems have been discussed. Equipment requirements have been discussed and examples of equipment meeting. those requirements which have been developed by Stromberg-Carlson have been described. A special tape transport has been described which has been developed to meet a particular requirement in data transmission systems, where it is desired to accumulate data at a slow asynchronous rate and deliver it at a rapid, synchronout rate so as to obtain full time usage of a telephone facility.

The general approach being taken at Stromberg-Carlson in the development of card to tape, tape to tape, and card to card systems is to arrive at a complete line of components which can be interconnected in flexible fashion to meet a variety of requirements for specific data transmission applications.

References:

1. J. L. Wheeler, "High-Speed Digital Data Transmission", IRE Fourth National Aero-Com Symposium, Rome, N.Y., October 22, 1958.

2. J. L. Wheeler, "Preparation, Transmission and Distribution of Information in an Automatic Data Communition System Utilizing Telephone Facilities", AIEE Empire District No. 1, 1959 Spring Meeting, Syracuse, N. Y., April 29, 1959.

3. F. David, C. J. Zarcone, R.L. Wolff, "Card-to-Magnetic-Tape Data System", AIEE Conference Paper CP60-917, 1960 Summer General Meeting, June 21, 1960.

4. J. L. Wheeler, "Telephone Line Input to an IBM 704 Computer", IRE Sixth Annual Communications Symposium, Utica, N. Y., October 4, 1960.

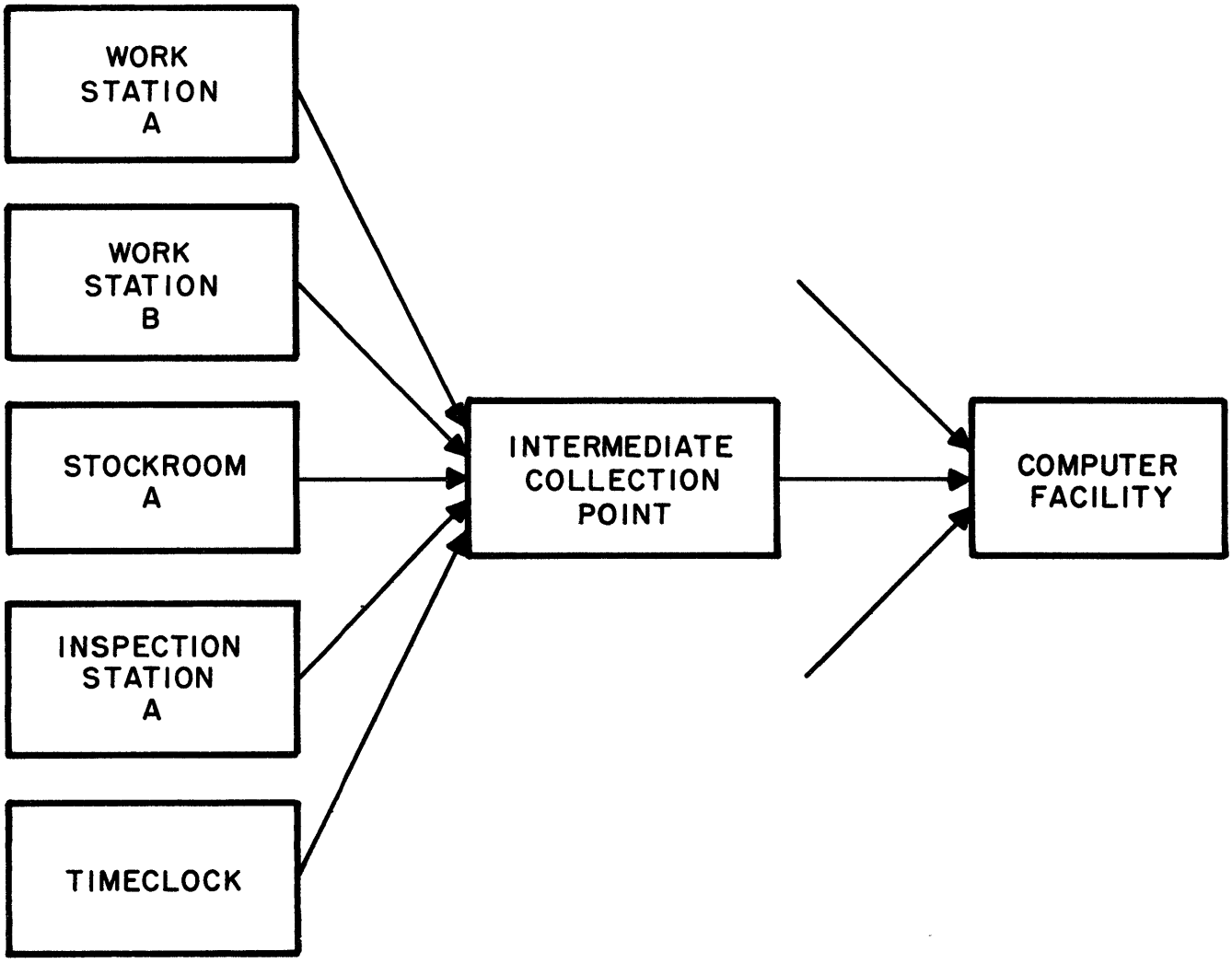
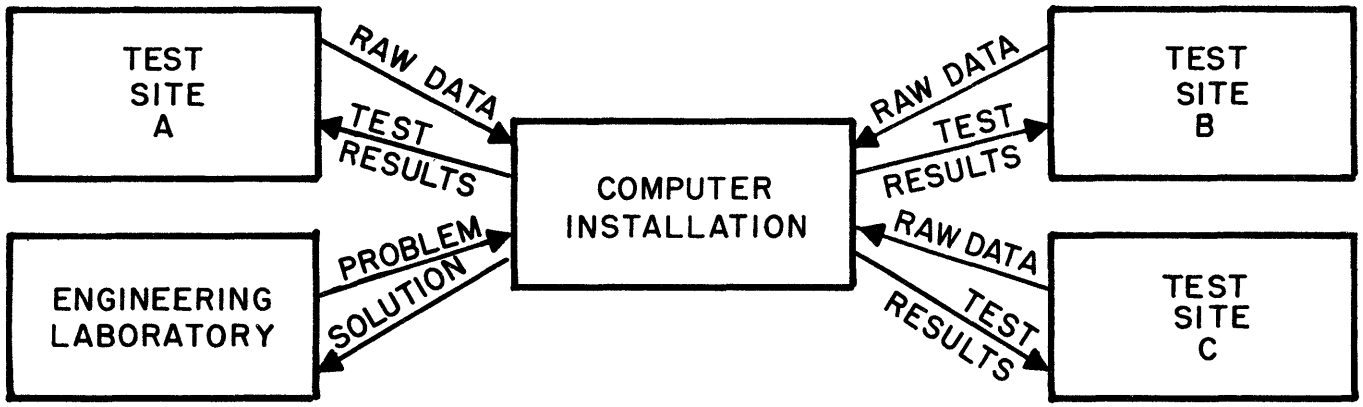
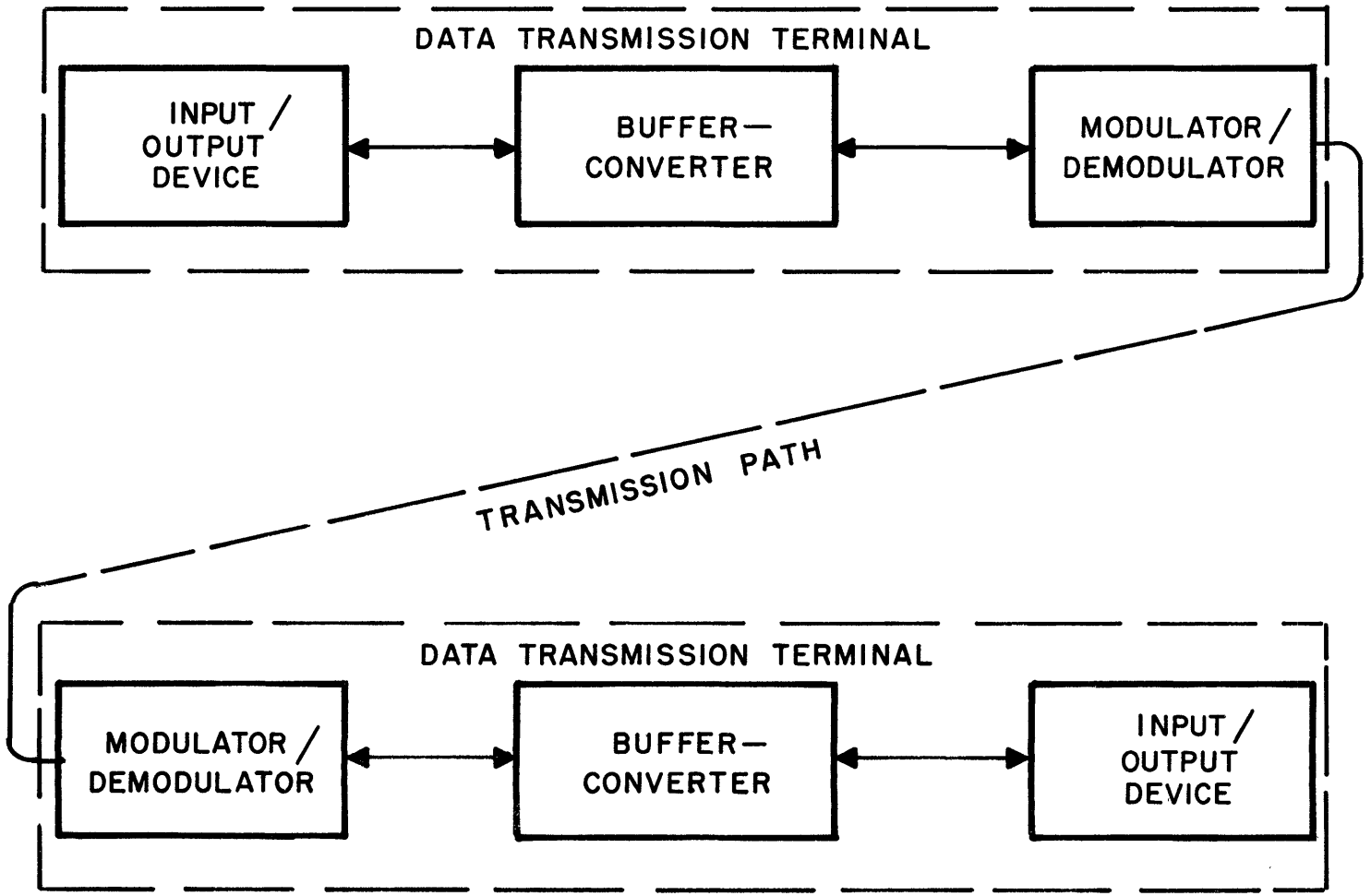


Fig. 1. Data Collecting System



SCIENTIFIC DATA

Fig. 2. Scientific Data



GENERALIZED DATA COMMUNICATION SYSTEM

Fig. 3. Generalized Data Communication System

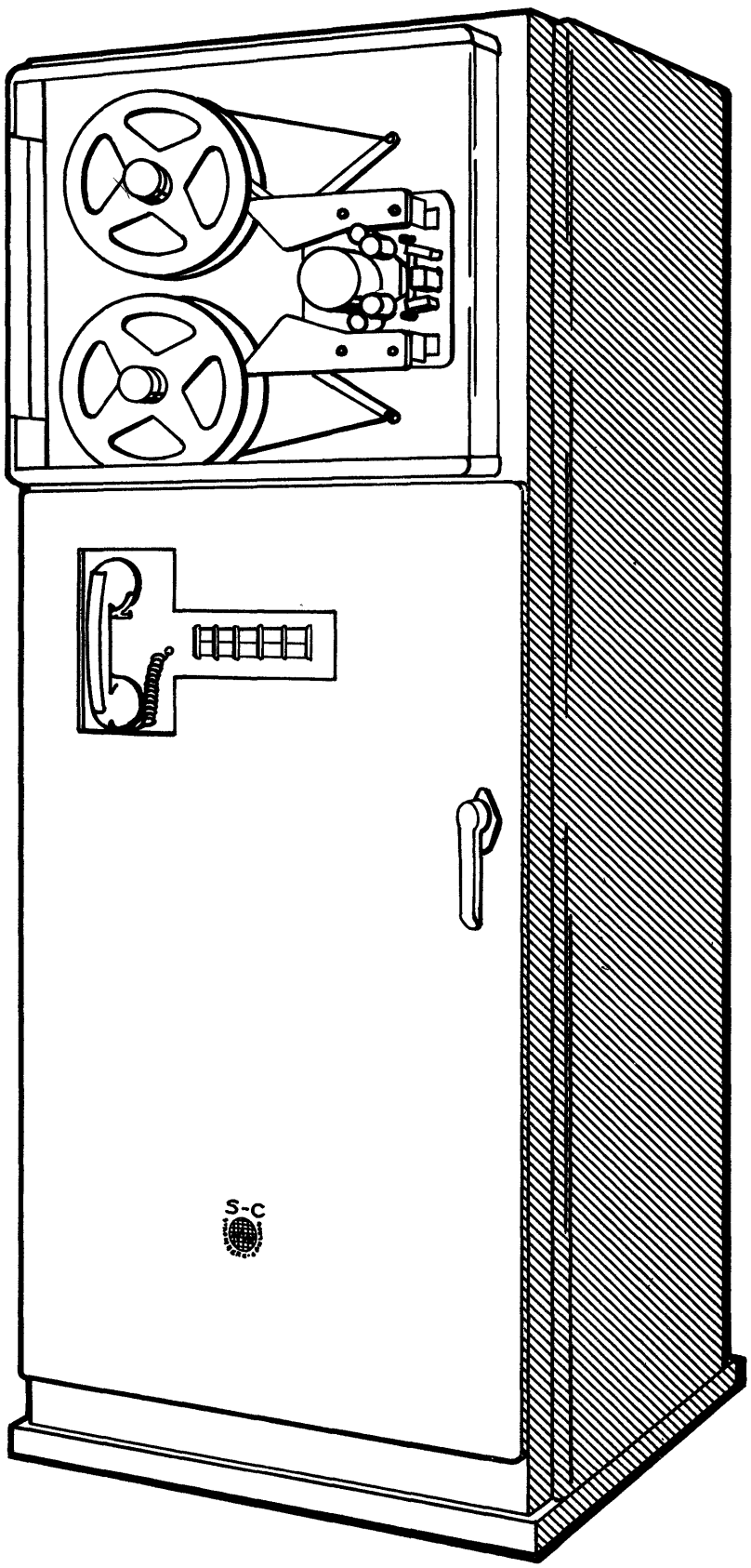


Fig. 4. Tape Transmission Terminal

Buffer-Converter Functions

Transmitting	Receiving
Parallel/Serial Conversion	Serial/Parallel Conversion
Generate SOR & EOR Characters	Check Lateral & Longitudinal Parity
Synchronize Data Transmission Rate and Tape Transport	Generate Retransmission Signal in Event of Errors
Detect End of File	Synchronize characters to incoming Data
Cause Data Record to be Retransmitted Upon Receiptof Error Signal from Receiver	Generate EOR Gap Detect End of File
Provides Intercom Capability	Provides Intercom Capability

Fig. 5. Buffer-Converter Functions

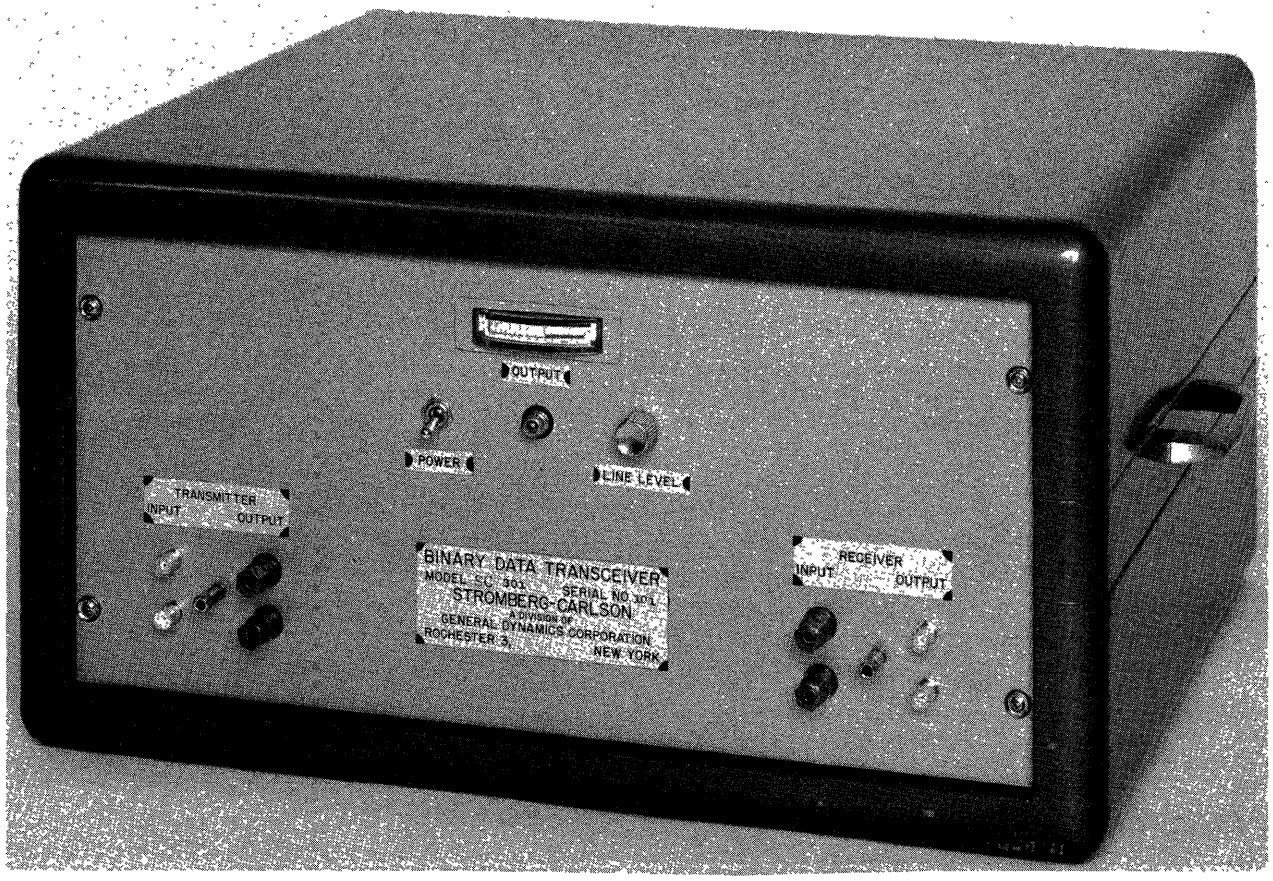


Fig. 6. SC-301 Binary Data Transceiver

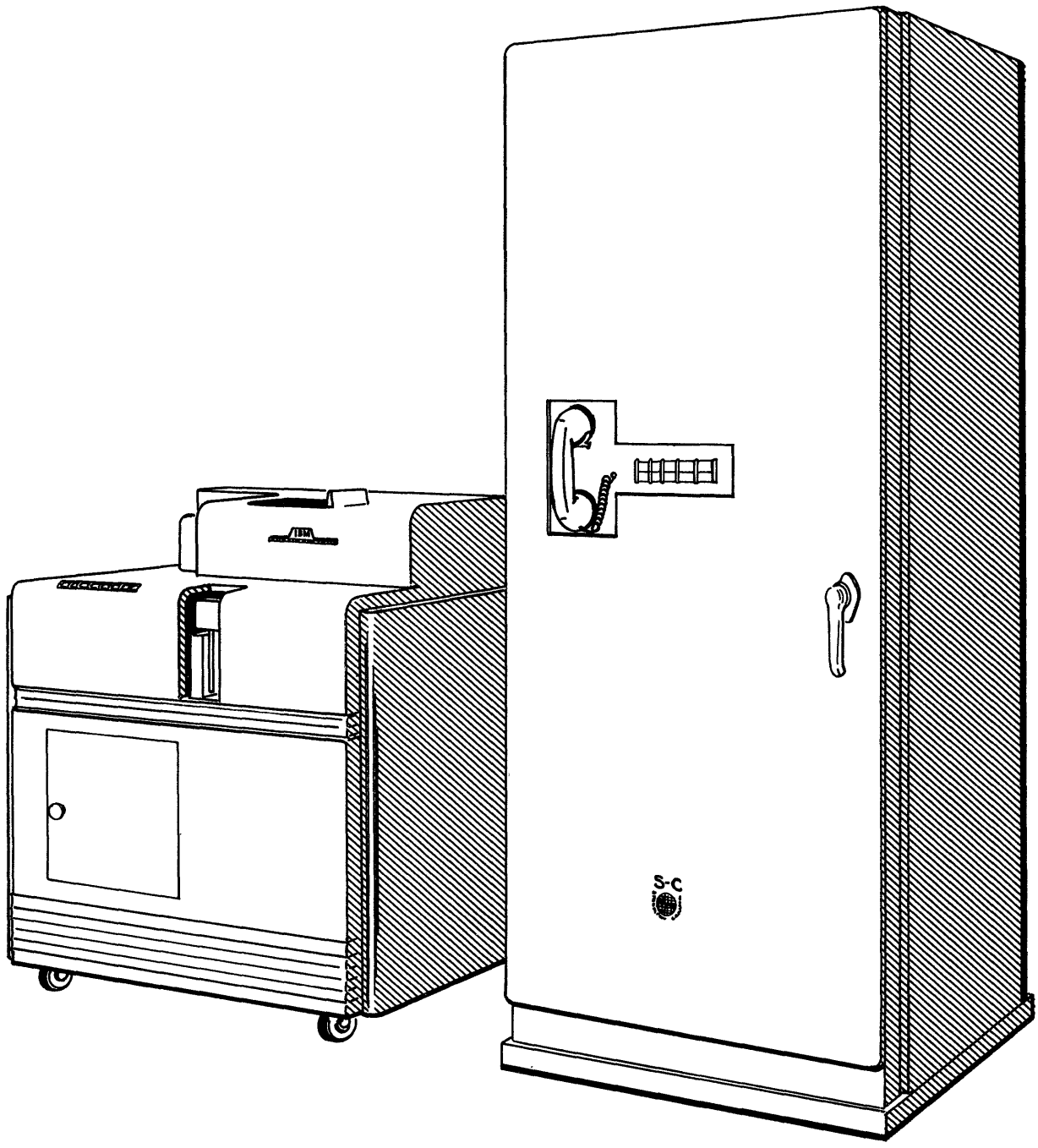


Fig. 7. Card Transmission Terminal

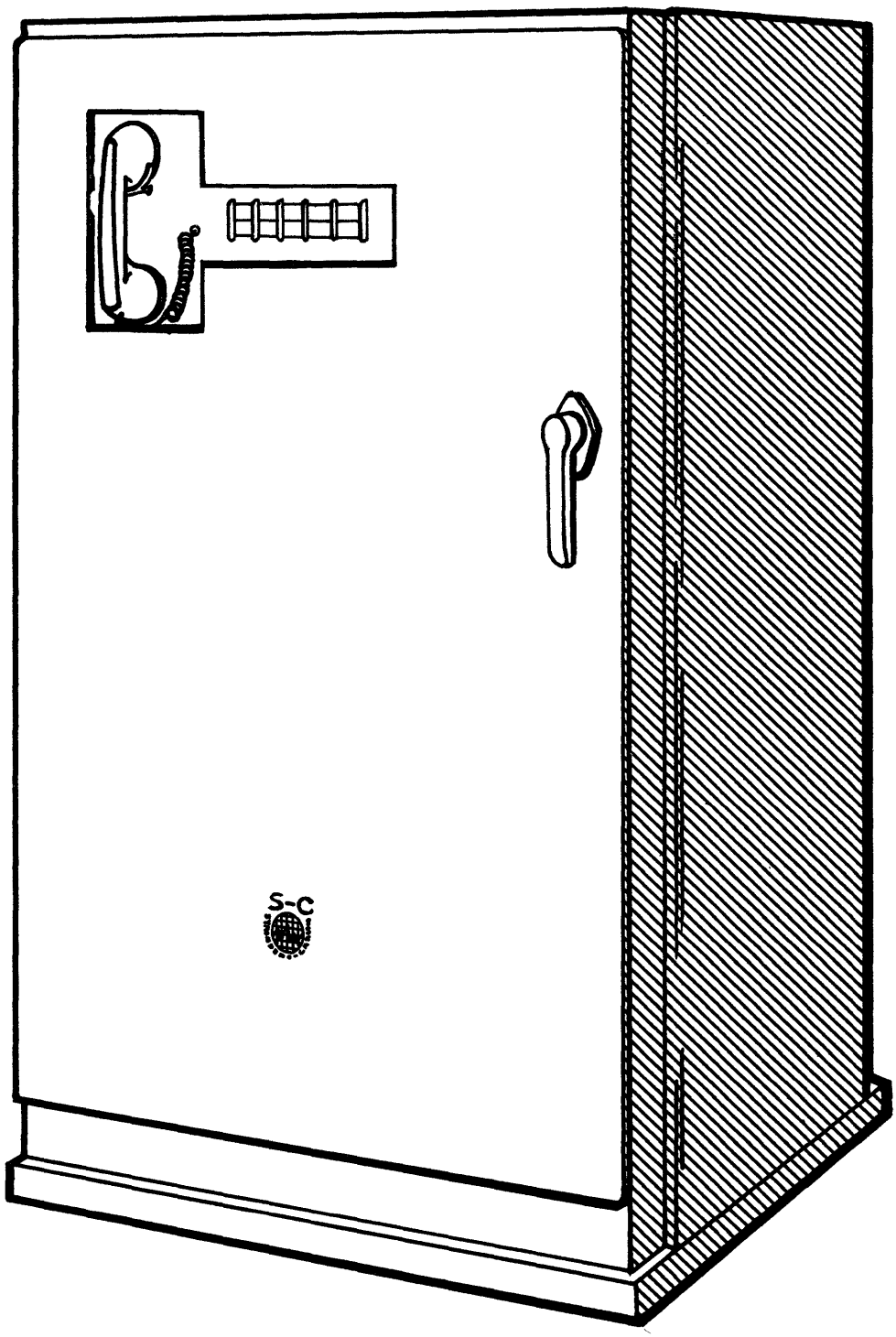


Fig. 8. Computer Transmission Terminal

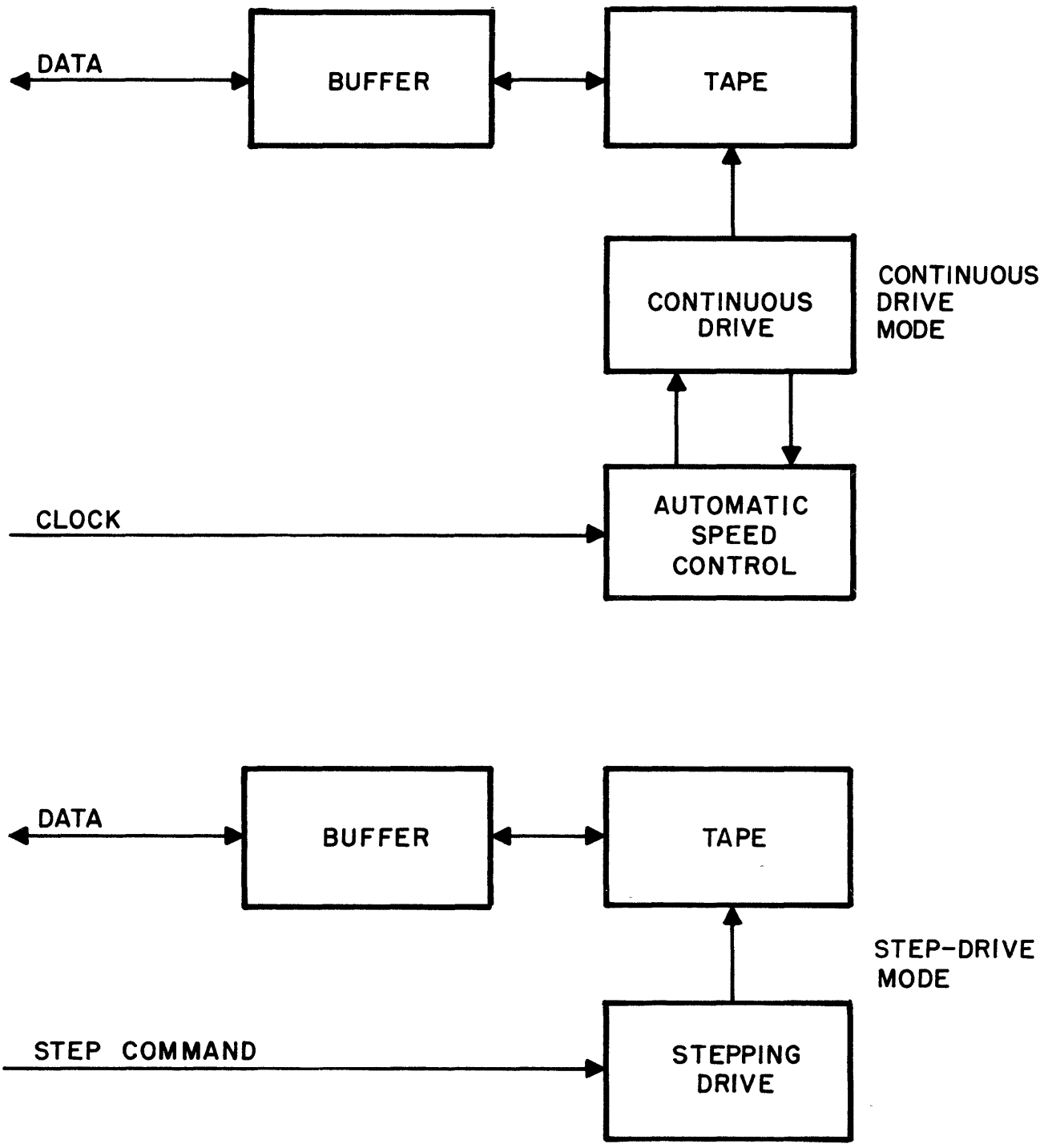


Fig. 9. Tape Transport Block Diagram

PARALLEL COMPUTING WITH VERTICAL DATA

William Shooman
System Development Corporation
Santa Monica, California

Summary

A novel technique called Vertical Data Processing (VDP) for the manipulation of data in digital computers is presented. Multiple data are processed simultaneously one bit at a time using Boolean operations. Many classes of problems appear adaptable to this technique.

A hypothetical VDP computer which embodies both VDP and conventional techniques is proposed and its advantages discussed.

Introduction

In the continuous quest for increasing the speed of Electronic Data Processing Machines, two distinct procedures are available. One is to improve the hardware technology; the other is to improve computer organization. This paper is concerned with the latter.

A novel technique for the simultaneous manipulation of multiple data in digital computers is presented. This technique is called Vertical Data Processing (VDP), in contrast to conventional methods which will be referred to as Horizontal Data Processing (HDP). Data organization for VDP is described and a VDP machine defined. It is shown that the time taken to perform VDP operations is not a function of the number of numbers being processed.

A VDP machine which processes vertical data is shown to have certain limitations which are eliminated by a hypothetical computer design. General specifications for the hypothetical machine (called the Orthogonal Computer) are given. Descriptions and algorithms for several VDP instructions (one of which is an add instruction) are given.

VDP logic is shown to be strikingly different from that of HDP. Masks play the role of decision functions with the result that there is virtually no branching in VDP. VDP is applied to the following specific problems; 1) an FICA computation, 2) finding the rank of a number in a sequence of numbers, and 3) the translation of mnemonic operation codes to their machine language representation. Time comparisons for VDP and HDP are made. For these problems VDP ranges from 32 to 660 times as fast as HDP. Finally a cost estimate for the Orthogonal Computer is presented and some possible input/output limitations are noted.

Data Organization for VDP

Suppose that we are given r numbers in memory. These numbers are expressed in the computer by a matrix of bits. In the usual matrix (UM) each of the r numbers resides in one computer word, or row of the matrix. We have investigated the advantages of expressing the numbers by a matrix other than the UM; that is, by the transpose of the UM (UMT). In the UMT each number resides in a column of the matrix; consequently, each of its bits is in a different computer word. Since the computer has direct access to computer words only, it is not possible to obtain directly one of the r numbers. Instead all r numbers are simultaneously processed one bit at a time by means of the logical operations, 'and', 'not', 'inclusive or', and 'exclusive or'. It is clear that Boolean functions can be developed to perform the usual operations of conventional computers (as, for example, has been done for serial computers).

The VDP Machine

A VDP machine will now be defined as a machine which has the hardware to perform computations directly by addressing the data orthogonally. A program simulating a VDP machine which processes data in UMT form was checked out on the 709. The set of simulated instructions included the arithmetic and logical operations usual to HDP machines.[1] The algorithm used for the simulated add instruction is now shown.

Add Algorithm and Timing

Given two sequences of numbers (A_j) and (B_j) $(1 \le j \le r)$, we want to compute the r sums $S_j = A_j + B_j$. For the sake of simplicity we will assume that each number consists of precisely n bits and is non-negative. Let $(a_{1,j}\ a_{2,j} \cdots a_{n,j})$, $(b_{1,j}\ b_{2,j} \cdots b_{n,j})$, and $(s_{0,j}\ s_{1,j} \cdots s_{n,j})$ be the binary representation of (A_j), (B_j), and (S_j) respectively. To compute S_j, we give an algorithm to compute $s_{i,j}$ $(i = n, n-1, \ldots 1)$.

Let

$$s_{i,j} = a_{i,j} \oplus (b_{i,j} \oplus c_{i,j}); \quad c_{n,j} = 0;$$

$$c_{i-1,j} = \left[(a_{i,j} \oplus b_{i,j}) \vee (b_{i,j} \oplus c_{i,j}) \right] \oplus s_{i,j};$$

$$s_{0,j} = c_{0,j}$$

[1] See Instruction List in Appendix.

Where ⊕ stands for exclusive or and ∨ for inclusive or.

It is clear that all r sums can be simultaneously computed using this algorithm, if r is equal to or less than the bit length of the computer word. In a VDP machine such an add is executed in $3n + 1$ memory accesses. The number of memory accesses is a function of n (bit length of the numbers being processed) only; in particular, it is not a function of r (the number of numbers). This important property is inherent in all VDP instructions and will be the basis for a hypothetical computer design.

Limitations of Data in UMT Form

One limitation of the simulated VDP machine is that the number of numbers that can be simultaneously processed is restricted to the bit length of the computer word. Another limitation is that data must be input in either the UMT form (that is, vertically) which creates some difficulty; or in UM form and then transposed. Also, data undergoing VDP is not suitable for HDP because of the data organization, although it is clear that some operations can be more efficiently performed in HDP (for example, summing two numbers). These limitations can be eliminated by a new computer design.

The Orthogonal Computer

Consider a hypothetical machine consisting of a conventional (HDP) machine of word length L, with the added capability of being vertically addressable in some limited region of memory; specifically in K nonoverlapping blocks, each block consisting of R consecutive memory registers. The vertical addressing is to be restricted so that an addressable column consists of precisely R bits and is contained in one of the blocks. Consequently there are exactly K times L addressable columns. The central processor is to contain a number of vertical registers (each register containing R flip-flops) and VDP hardware relating these registers to the K blocks. While a computation is being performed in some of the K blocks, input-output (I/O) may be going on in other blocks. If data are input to a subset of the K blocks in UM form, processing in parallel by means of vertical addressing satisfies the definition for a VDP machine. This machine will be referred to as the Orthogonal Computer.

Timing for an Add in the Orthogonal Computer

Suppose we have two sequences of numbers (A_i) and (B_i) $(1 \leq i \leq R)$, each number of bit length n and non-negative, and we want to compute the R sums $S_i = A_i + B_i$. Let (A_i) and (B_i) be in one or more of the K blocks (depending on the ratio of n to L) in UM form. Let $(a_{i,1}\ a_{i,2} \cdots a_{i,n})$, $(b_{i,1}\ b_{i,2} \cdots b_{i,n})$, and $(s_{i,0}\ s_{i,1} \cdots s_{i,n})$ be the binary representation of (A_i), (B_i), and (S_i) respectively. Using vertical addressing and the indicated change in subscript notation, it is clear that the simulated add algorithm presented above can generate the R sums (S_i) in a subset of the K blocks. The number of memory accesses is still $3n + 1$. In HDP, 8 times R memory accesses are generally required to obtain the S_i. Consequently, for this type of add instruction, with R equal to (on the order of) 1000, the Orthogonal Computer ranges from 55 (n = 48) to 500 (n = 5) times as fast as HDP. It is to be noted that both the VDP and HDP mode may be used on the same data in the Orthogonal Computer since data are input in UM form.

VDP Logic

VDP and HDP differ strikingly in their overall logical flow. THERE IS ESSENTIALLY NO BRANCHING IN VDP. Let us consider a branch point in an HDP program. Suppose that r numbers (A_i) $1 \leq i \leq r$ are being processed. Let the branch point partition the r numbers into two classes C_1 and C_2, (k numbers in C_1 and therefore $r - k$ in C_2) such that if A_i is in C_1, then A_i is to be processed by computation T_1, and similarly if A_i is in C_2, then A_i by T_2. Figure 1 shows the branch in HDP. In HDP the question must be asked for each A_i and correspondingly one of the computations T_1 or T_2 performed. Assuming that as many as r numbers can be simultaneously processed by VDP, a method is now described whereby only the results of computation T_1 on the numbers in C_1 and of T_2 on the numbers in C_2 are obtained.

Masking

Whenever a bit is written in memory during any VDP operation, the bit must pass through a gate. A gate is represented by a zero or one. A gate is open (allowing pass through) if its representative is a one. If its representative is a zero, the corresponding bit location in memory is left undisturbed. A string of representatives (b_i) $1 \leq i \leq r$ is called a mask. At that point in VDP corresponding to the branch point in HDP, a mask is generated such that b_i is a one, if and only if the answer to the question for A_i is yes. T_1 is now performed on all r numbers using the mask, which represents r gates (k of which are open). Consequently, only results of T_1 on numbers in C_1 are written in memory. The mask is then complemented, resulting in a mask of r gates ($r - k$ of which are open). This mask is used similarly in the T_2 computation on all r numbers, so that only results of T_2 on numbers in C_2 are written. Figure 2 shows the sequence of VDP operations.

'Compare' Description and Algorithm

A set of instructions called 'compare' plays a significant role in VDP on various levels. A description of 'compare >' and its algorithm is now given.

We are given r + 1 numbers consisting of a sequence $(A_i) 1 \leq i \leq r$, and a constant C. We again suppose that each number is non-negative and of bit length n. Let C and the (A_i) be represented in binary form. Let $C = C_1C_2 \ldots C_n$, and $A_i = a_{i,1} a_{i,2} \ldots a_{i,n} (1 \leq i \leq r)$. Let $C' = C_1C_2 \ldots C_t (t \leq n)$, where C_t is the least significant zero bit of C. 'Compare >' is to generate r bits (b_i), such that b_i is a one, if and only if $A_i > C$. To compute the r bits, we give the algorithm to compute the i^{th} bit. Let $b_i =$

$$a_{i,1} *1 \left\{a_{i,2} *2 \left[\ldots (a_{i,t-1} *_{t-1} a_{i,t}) \ldots \right] \right\},$$

using C' as follows. If the j^{th} bit of C' is a zero $(1 \leq j \leq t-1)$, $*_j$ is interpreted as inclusive or; if it is a one, $*_j$ is interpreted as and. Since the data is addressed orthogonally, it is clear that all $b_i (1 \leq i \leq r)$ can be generated simultaneously, provided that r is not too large. The number of memory accesses needed to execute 'compare >' in a VDP machine is t + 1 (t is the bit length of C').

'Compare >' Used as Mask Generator

The following example shows how 'compare >' is used efficiently in generating masks to be used as decision functions. Suppose that we are computing the FICA deductions in a payroll program. For each payroll period, 3% of total income is deducted and accumulated for FICA until the accumulated total is equal to \$144. Let r accumulated totals and \$143.99 play the respective roles of the $(A_i) 1 \leq i \leq r$ and C in the 'compare>' algorithm above. A sequence of r bits is generated such that the i^{th} bit is a one if and only if the i^{th} accumulated total is greater than \$143.99. This sequence of bits is complemented and then used as the mask in the incrementing computation. Consequently, only those totals that are less than \$144 are incremented. \$143.99 is represented by 14 bits, the least significant six of which are ones. Therefore t = 8 for this 'compare >' and we need 9 memory accesses in order to execute the instruction. For this problem, each time 'compare >' is performed in VDP, r comparisons would be necessary in HDP. About 6 memory accesses are generally needed to perform a comparison. For this function, the Orthogonal Computer is approximately 660 times as fast as HDP for r = R = 1000.

'Compare ≥' Used to Compute Rank

Another interesting use for 'compare' is given by the following example. We are given a sequence of r numbers $(A_i) 1 \leq i \leq r$, non-negative and of bit length n. We want to find the number (k) of numbers in (A_i) that are less than a particular number A_j in the sequence. This number k is frequently referred to as the rank of A_j in the sequence (A_i). To obtain the rank of A_j, we let the (A_i) and A_j play the respective roles of the (A_i) and C in the 'compare >' algorithm, with an exception. Let $a_1a_2 \ldots a_n$ be the binary representation of A_j. Let $A_j' = a_1a_2 \ldots a_t$ $(t \leq n)$, where a_t is the least significant one[2] bit of A_j. This change results in a 'compare ≥' rather than 'compare >'. The generated r bits are then complemented obtaining r bits such that the i^{th} bit is a one, if and only if $A_i < A_j$. These r bits therefore consist of exactly k one bits and r - k zero bits. The execution time for an instruction giving the count of the number of one bits in a given register should be equivalent to about two memory accesses. Consequently, the time to compute rank in the hypothetical machine should be approximately equivalent to t + 3 memory accesses. In HDP, if the (A_i) are not sorted, r - 1 comparisons are necessary to compute rank. Assuming that r = R = 1000, the Orthogonal Computer ranges from more than 115 (t = 48) to about 635 (t = 5) times as fast as HDP in computing rank in an unsorted sequence.

VDP Applied to Compilers

Every compiler has within it an assembly process. Let us suppose that a given machine has N operation (op) codes. Let R mnemonic instructions be in one of the K blocks of the Orthogonal Computer. The assembler must translate the instruction list from the menemonic to machine language. The result should be R machine language instructions in another of the K blocks. Let O_j $(1 \leq j \leq N)$ be the binary representation of the j^{th} mnemonic op code. For each j, we perform 'compare =', comparing O_j against all R op codes. The j^{th} 'compare =' generates a mask (b_i) $1 \leq i \leq R$, such that b_i is a one, if and only if the op code of the i^{th} mnemonic instruction is equal to O_j. Using the machine language representation of O_j as a constant, we perform 'constant insert' on

[2] In the 'compare >' algorithm, C_t of C' is the least significant zero bit of C.

all R op codes using the mask. The 'constant insert' instruction duplicates a constant, R times in a given block. Consequently, because of the mask, only those op codes that are equal to O_j are translated to the machine language representation of O_j. After N 'compare =' and N 'constant insert' are performed, each of the R mnemonic op codes will have been translated into its corresponding machine language representation.

If the mnemonic op code consists of three 6-bit characters, and the corresponding machine language representation requires twelve bits (for example, the 7090), then the number of memory accesses required in VDP for the op code translation is 32N. In HDP, a search is performed in a table of N op codes, with an approximate average of $\log_2 N$ comparisons for each op code. Approximately $8(\log_2 N) + 15$ memory accesses are needed to find and translate each op code. For N = 60, and R = 1000, the Orthogonal Computer is about 32 times as fast as HDP.

Storage Economy by Packing Data

We order the K blocks of the Orthogonal Computer, and also the L columns in each block from most to least significant bit. We now linearly order all KL addressable columns by demanding that the last column in any block (excepting the last block) directly precede the first column of the next block. We can now think of these KL columns as a matrix of R rows and KL columns. Suppose that we have t sequences of data to process (A_i^j) $(1 \le i \le R;\ 1 \le j \le t)$. For each j, let n_j be the bit length of the data in (A_i^j). Let $N = \sum_{j=1}^{t} n_j$. It is clear that no generality is lost in VDP, if the t sequences are packed into any consecutive N of the KL columns, provided that N is not too large.

Cost of Orthogonal Computer

Preliminary investigation has begun in estimating the cost of the additional capabilities of the Orthogonal Computer. The results are now presented.

A memory consisting of 2^{15} core locations, each of bit length 48, was arbitrarily assigned to the HDP computer which was used as a base for the Orthogonal Computer. The estimate is that an additional 35% to 45% of the cost of the HDP computer would be incurred for R = 512 and K = 16.

I/O Limitations[3]

Computing is so fast using VDP, that input/output limitations may be an acute problem for the Orthogonal Computer. If K were as small as 3 (with a reasonable R), there would be few problems for which the machine would not be severely I/O limited. Flexibility with which to combat the I/O problem increases with increasing K, for fixed R; so does the cost of the machine. As R increases, VDP computing time remains constant, but I/O time per block and the cost of the machine increases. These are some of the considerations involved in choosing R and K.

Acknowledgements

This paper is the outgrowth of an idea originally conceived by Gerald Fine of the System Development Corporation. To my knowledge, Mr. Fine was the first to foresee the possibilities of organizing data vertically. He initiated the VDP project at SDC a little over a year ago. Thomas N. Hibbard, who was a member of the project interested me in VDP. Mr. Hibbard also did some of the early development work. It is a pleasure to acknowledge my indebtedness to these two individuals.

The author also wishes to thank the members of the Staff of the Center for Research in System Sciences at SDC for their general assistance; in particular, J. N. A. Hawkins who consulted on the cost and engineering aspects of the Orthogonal Computer, and Richard Brouse, Donald P. Estavan, and Seymour Ginsburg for their constructive criticism.

Appendix

The Orthogonal Computer Instruction

A typical instruction consists of an op code, three addresses (A_i), and three parameters (P_i). P_i specifies the bit length of the corresponding operand at A_i. Each A_i refers to one of the KL addressable columns, to one of several vertical flip-flop registers, or to a specified horizontal register of flip-flops, say, the HDP accumulator[4]. Some of the instructions do not use all the fields.

Instruction List	Timing[5]
Arithmetic	
Add Add magnitude Substract Substract magnitude	$P_1 + P_2 + P_3$

[3] A computer is said to be I/O limited, if a significant portion of machine time is spent waiting (i.e., not computing), while input/output is being performed.

[4] The HDP accumulator holds the constant used in the 'compare' instructions; it also permits adding, substracting, multiplying, and dividing the numbers of a block by a constant.

[5] The approximate number of memory accesses exclusive of those necessary to fetch the instruction.

Instruction List (continued)	Timing
Multiply Multiply magnitude Multiply and accumulate Divide Divide magnitude	$P_1\left(\frac{3P_2 + P_3}{4} + 1\right)$
Compare	
Compare greater than Compare equal Compare equal to or greater than	The larger of P_1 and P_2
Store	
Move column	2
Constant insert	2 P_1
Logical	
And	2
Inclusive or	2
Exclusive or	2
One's complement	2
Count ones	1

HDP

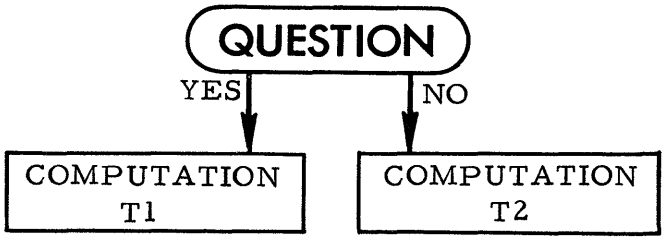


FIGURE 1

VDP

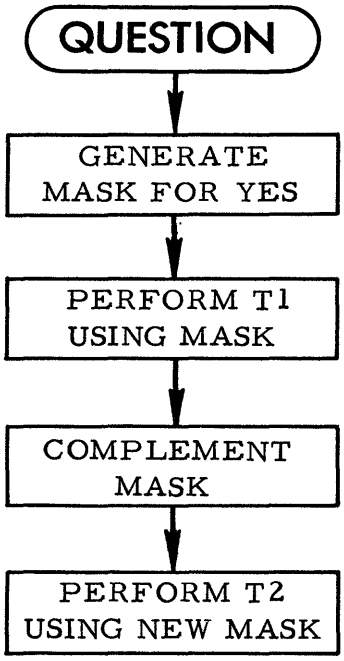


FIGURE 2

TABSOL
A FUNDAMENTAL CONCEPT FOR SYSTEMS-ORIENTED LANGUAGES

T. F. Kavanagh

Manufacturing Services
General Electric Company
New York, New York

Summary

Lack of efficient methods for thinking-through and recording the logic of complex information systems has been a major obstacle to the effective use of computers in manufacturing businesses. To supply this need, this paper introduces and describes "decision structure tables," the essential element in TABSOL, a tabular systems-oriented language developed in the General Electric Company. Decision structure tables can be used to describe complicated, multi-variable, multi-result decision systems. Various approaches to the automatic computer solution of structure tables are presented. Some benefits which have been observed in applying this language concept are also discussed. Decision structure tables appear broadly applicable in information systems design.

In addition, they are of interest because they revise many earlier notions on problem formulation and systems analysis technique. Decision structure tables will be an available feature in GECOM, General Electric's new General Compiler, which will be first implemented on the GE 225.

Introduction

Progress in computers can be broadly divided into two categories. First there is the work that essentially accepts computers for what they are, and directs its energies toward further refinement of the original hardware, and operating technique. Research to improve recording density on magnetic tape would certainly fit this description. In the second category are the efforts to advance by developing new areas of application. This latter work is directed toward generalizing the concepts and hardware, so that they apply to an ever-increasing span of problems and situations. Obviously, both groups are vital; but it was this second stimulus -- the desire to expand the area of economic application -- which motivated the research reported in this paper. While the earliest beginnings can be traced as far back as June, 1955, the primary research effort started in November, 1957, under the title of the Integrated Systems Project. Leadership was assigned to Production Control Service, a component in General Electric's Manufacturing Services. The basic purpose of the Project was to probe the potential for automating the flow of information and material in an integrated business system.

Then, as now, computers were making significant contributions in many areas. Unfortunately, one of these areas was not, as some would have it, in the operation and control of manufacturing businesses. Important advances were made in specific applications such as order processing payroll, and inventory record-keeping; but these represented only a small percentage of the total information processing and decision-making in even the smallest manufacturing firm. Still these early successes were very important. They developed confidence in computer performance and reliability; but even more, they encouraged systems engineers and procedures personnel to continue computer applications research. Similarly, management, under growing foreign and domestic competition, rising costs, and a seeming explosion in paperwork requirements, saw intuitively -- or perhaps hopefully -- that computers offered a possible approach to improved productivity, lower costs and sharply reduced cycle times. It was in this environment that the Integrated Systems Project began a comprehensive study of the decision-making and the information and material processing required to transform customer orders into finished products -- a major part of the total business system for a manufacturing firm.

The Decision-Making Problem

Once underway, it was soon apparent

that there was an enormous amount of decision-making required to operate a business. Indeed, the number and complexity of these decisions is perhaps the most widely underestimated and misunderstood characteristic of industrial information systems today. Tens-of-thousands of elementary decisions are made in the typical manufacturing business each working day. All are necessary to guide and control the many functional activities required to design products, purchase raw material, manufacture parts, assemble products, ship and bill orders, and so on. The typical factory is a veritable beehive of decisions and decision-makers; for example:

- "What size fuses shall we use on this order for XYZ Company?"-- a product engineer's decision.
- "What is the time standard for winding this armature coil?"- a manufacturing engineer's decision.
- "What test voltages shall be applied?"-- a quality control planner's decision.
- "What should be the delivery promise on this customer's order?"-- a production control planner's decision.
- "How much will this model cost."-- an accountant's decision.

This list of elementary day-to-day decisions could be expanded to cover all business activities. If this were done, the list would cover hundreds of sheets of paper before each activity listed all the decisions for which it was responsible. Moreover, some of these decisions are repeated many times each day for various sets of conditions. In the end result, one cannot help but be impressed with the multiplicity of these detailed choices and selections. But more importantly, making these decisions costs money, in many cases more money than the direct labor required to make the product. In addition, business performance is greatly affected by the speed and accuracy with which this decision-making is carried out.

Composing a detailed list of these elementary business decisions is more than an academic exercise. For one thing, such an analysis of an actual operating business will demonstrate conclusively that these elementary decisions are handled quite rationally (which is somewhat contrary to popular opinion.) One must be careful not to be misled by quick, superficial explanations which gloss over fundamental reasoning. In our present-day manual systems which emphasize files of quick answers, the logic behind the decision is often left unrecorded. As a result it is easy to lose contact with their rational nature, and frequently we tend to feel these decisions are substantially more intuitive than is actually the case. At times, some persistent as well as penetrating analysis (often through extensive interviewing of the operating personnel presently on the job) is required to uncover the true parameters and relationships on which operating decisions are really based. This arduous work is more than justified, for it establishes a sound conceptual foundation for automation, and hence the practical application of the concepts and techniques developed in this paper. Thus, once it is established that these operating decisions are rational, it should follow that they can be structured in a consistent logical framework. Such a structure is presented in this paper.

Operating vs. Planning Decisions

At this point let us define terminology a little more precisely, and stress that we are speaking about the detailed, elementary decisions required to "operate" a business as opposed to "planning" one. First, a decision in its simplest form consists of selecting one unique alternative from an allowed set of possible actions. Operating decisions are defined in the context of this paper as selecting the appropriate course of action in accordance with given problem conditions to operate the business successfully. Operating decisions may be assumed to be made under "conditions of certainty." The solution for a specific set of problem conditions will always be the same. Under these premises, the action or outcome decided on can always be predicted. In a pragmatic sense, the decision-making process may be classed as "causal"; that is, B may be said to follow from A. For example, an engineer's decision to install fuses might follow from a customer's requirement for independent circuit protection.

The relevant factors or parameters affecting the decision can also be determined. The relationship values are known. For example, in most homes, the current carrying capacity of the house wiring is the only parameter value one needs to know to select an appropriate fuse. In an industrial application, however, the values of at least three additional parameters are usually required: voltage, time and type of fuse mounting. The strategy and the alternate outcomes are known; that is, the per-

missible fuses are known. To continue the illustration, the fuse selection may be limited to those carried in the stockroom; otherwise the bounds of the operating decision system are exceeded and the decision-maker would appeal to a higher authority.

To approach the analysis of operating decisions from another viewpoint, it might be compared to a linear programming problem, and as will become evident, a linear programming solution might be considered as somewhat of a mathematical bound for the class of decision-making systems under discussion.

These operating decisions are quite apart from the planning decisions of a business. The "planning", "administrative", or "policy" decisions in a business are basically those prior commitments which permitted all the assumptions about operating decision systems in the preceding paragraphs (i.e. certainty, causality, known relationships, etc.) Some examples of planning decisions are:

- "Shall fuses, circuit breakers, or both be used on the product line?"-- a product engineer's planning decision.
- "Should this group of parts be made on the screw machine or from die castings?"-- a manufacturing engineer's planning decision.
- "Should this component be inspected before or after the milling operation?"-- a quality control planning decision.
- "What rule shall be used to determine the correct order quantity?"-- a production control planner's decision.
- "What is an appropriate cost-of-money?"-- an accountant's planning decision.

These are typical planning decisions made in designing an operating decision system. To make the distinction clear, consider the design engineer who is motivated by cost considerations to put fuses on the economy part of the product line, while specifying circuit breakers on more deluxe models. Or consider the production control planner who selects one of the common square root formulas for determining all order quantities. Once he puts this decision rule in the operating system, order quantities for every part will be determined using this square root formula with specific values for cost, lead time, usage shelf life, etc., appropriate to the specific item being ordered. Assuming the operating decision system is automatic, and this is the intention, the production control planner need not make any order quantity determinations himself. Rather he will be watching the measures of operating system performance (inventory level, number of shortages, ordering costs, etc.) to see how well his decision rule is working. Incidentally, it's worth noting that the production control systems designer will be using a "cost-of-money" figure supplied by accountants and an annual requirements figure projected by salesmen. Of course, the objective of this fundamental decision analysis is to suggest a conceptual scheme which will permit automating all the routine operating decision-making required to direct a business, thus permitting the engineers, planners, and other technical advisors, to concentrate on doing a better job in design.

Specifying Decision Systems

But great difficulties still remain. As already pointed out, operating decision systems are invariably large and complex, containing multi-variable, multi-result decision problems with sequence of solution difficulties thrown in on the side. One serious problem which arises quickly is the actual development of the decision logic itself. Numerous techniques have been proposed ranging from precise, legalistic verbal statements to complex mathematical equations. Among these however, it appears that matrix-type displays and flow charts are the most common. The matrix-type displays appear under a variety of names: collation charts, tabulated drawings, standard time data sheets, etc. For example, engineers have frequently used collation charts to show direct relationships between end-product catalog numbers and component identification numbers. Typically, however, collation charts are a tabulation of past decisions rather than a description of the logic used to derive them. Matrix-type displays often suffer from redundancy and frequently become large and unwieldy as operating tools. Similarly, they make no allowance to sequential decision-making.

Flow charts handle this sequence problem very nicely. This graphic method describes a decision system by the extensive use of symbols for "mapping" the various operations. A variety of flow chart techniques are used in factory methods and office procedures work.

They are particularly effective in relatively straightforward, sequential decision chains but run into difficulty when describing multi-variable, multi-result decision processes. As an illustration, flow charts have been used extensively to document the detailed logic of computer programs; but some harried computer programming supervisors still maintain that the best way to transfer program knowledge is to reprogram the job. The difficulty of interpreting someone else's flow charts is certainly one of the major trials in today's computer technology.

In addition to these more popular tools numerous other diagramming or charting techniques have been useful in limited problem areas. However, the basic problem remained: there was really no effective, uniform method for thinking about and specifying decision systems as complex as those required to operate a business. To help solve this problem, the Integrated Systems Project developed a new technique which combines key characteristics of earlier methods and adds some new features of its own. This new technique is called the decision structure table. The balance of this paper will describe what decision structure tables are, how they work, and the results of their use in General Electric.

Structure Table Fundamentals

Structure tables provide a standard method for unambiguously describing complex, multi-variable, multi-result decision systems. Thus, each structure table becomes a precise statement of both the logical and quantitative relationships supporting that particular elementary decision. It is written by the functional specialist in terms of the criteria or parameters affecting the decision and the various outcomes which may result.

A structure table consists of a rectangular array of terms, or blocks, which is further subdivided into four quadrants, as shown in Figure 1. The vertical double line separates the decision logic on the left from the result functions or actions which appear on the right. The horizontal double line separates the structure table column headings or parameters above from the table values recorded in the horizontal rows below. Thus, the upper left quadrant becomes decision logic column headings, and is used to record, on a one per column basis, the names of the parameters (P_{0j}) effecting the decisions. The lower left quadrant records test values (p_{ij}) on a one per row basis, which the decision parameter identified in the column heading may have in a given problem situation. The upper right hand quadrant records the names of result functions or actions to be performed (R_{0j}) as a result of making the decision, once again on a one per column basis. Similarly the lower right quadrant shows the specific result values (r_{ij}) which pertain, directly opposite the appropriate set of decision parameter values. Thus, one horizontal row completely and independently describes all the values for one decision situation.

There is, of course, no limit to the number of columns (decision parameters and result functions) in any given structure table. Even the degenerate case where the number of decision parameters goes to zero is permissible. Also there is no limit on the number of decision situations (rows). Thus, the dimensions (columns by rows) of any specific structure table are completely flexible, and are a natural outgrowth of the specific decision being described. A series of these structure tables taken in combination is said to describe a decision system.

Rather than become further involved in abstract notation, let's consider some actual illustrations to develop an insight into the nature of structure tables. For example, the oversimplified illustrative structure table in Figure 2 states that an elementary decision on transportation from New York to Boston in the afternoon is (according to the person who developed the decision logic) a function of three decision parameters: Weather, Plane Space, and Hotel Room. Weather has only two value states, Fair or Foul; Place Space is either OK or Sorry; and Hotel Room can be Open or Filled. In terms of results, Plane or Train are the only permissible means of Transportation. Following the illustrative problem, we see by inspection that the solution appears in the second row. Therefore, Train is the correct value for Transportation, Other Instructions are Cancel Plane, and this is the End of the decision problem.

The intent of this simple structure table is to provide a general solution to this particular decision situation, and if the problem of afternoon trips to Boston ever arises (and one assumes that it frequently does), then an operating decision can quickly be made by supplying the current value of Weather, Plane Space, and Hotel Room, and, of course, solving the structure table. Solving a structure table consists of examining the specific values assigned the decision parameters in the problem statement and comparing or "testing" these values against

the sets of decision parameter values recorded in the structure table rows. Testing proceeds column by column from the first decision parameter to the last (left to right) and thence row by row (top to bottom). If all tests in a row are satisfied, then the solution is said to be in that row and the correct result values appear in the same horizontal row directly opposite to the right of the double line. When a test is not satisfied, the next condition row is examined.

When a particular structure table has been solved, it is often necessary to make more decisions. To specify what decision is to be made next, the last result column of the structure table may be assigned as a director to provide a link to the next structure table. Notice the last row in the illustrative structure table which specifies that for any value of Weather, with no Plane Space, and no Hotel Room, the decision-maker is directed to solve the next structure table, Transportation, New York-Boston, a.m. -- which is another structure table describing how to select a means of transportation in the morning.

In a similar fashion, the systems designer would use a whole system of structure tables to describe a more realistic operating decision problem. He completely controls the contents of each table, as well as its position in the sequence of total problem solution. He may decide to skip tables, or, if desired, he may resolve tables to achieve the effect of iteration. In any event, the entire system of tables, just as each individual structure table, will be solved using specific decision parameter values appearing in the problem statement. In other words, solving a set of structure tables consists essentially in re-applying the systems designer's operating decision logic.

Having completed this quick and very simplified introduction to structure tables, let us now return to consider each structure table element in greater detail. This will provide a deeper insight into the power of the structure table technique, as well as a better understanding of how they are used to describe operating decision systems. The illustrations are drawn from actual operating decision problems.

Structure Table Tests

Comparisons or tests between problem parameter values (pv) and decision parameter test values (tv) need not be simple identities, such as those used in the previous illustration. Actually the problem parameter values may be compared to the decision test values in any one of the following ways in any structure table block:

EQ	pv = tv	problem value is equal to test value.
GR	pv > tv	problem value is greater than test value.
LS	pv < tv	problem value is less than test value.
NEQ	pv ≠ tv	problem value is not equal to test value.
GREQ	pv ≥ tv	problem value is greater than or equal to test value.
LSEQ	pv ≤ tv	problem value is less than or equal to test value.

This broad selection of test types (or relational operators as they are known technically) greatly increases the power of individual structure tables and sharply reduces size. It permits testing limits or ranges of values rather than only discrete numbers. In Figure 3, TABLE 1000 uses several difference test types to bracket continuous and discontinuous intervals. Also note in Figure 3, that the relational operator may be placed in the test block immediately preceding the test value, or in the column heading immediately following the decision parameter name. When this latter notation is used, the relational operator in the column heading applies to all test values appearing immediately below.

Test values are not limited to specific numbers on alphanumeric constants (indicated by quotation marks); a test block may also refer to the contents of any name. In this case of course, the current contents of that named field are compared to the problem parameter value in accordance with the test type. For example, TABLE 1005 in Figure 3 tests the current value of INSUL~TEMP against MAX~TEMP to make certain that insulation temperature ratings are satisfactory.

In addition to these simple comparisons it is also possible to formulate compound structure table blocks involving two decision parameters or test values using a relational or logical operator.

The following logical operators may be used:

OR	tv_1 OR tv_2	first test value or the second test value.

AND	pv_1 AND pv_2	first problem value and second problem value.
NOT	tv_1 NOT tv_2	first test value and not second test value.

Also the truth or falseness of a compound decision parameter or test value statement can be tested with the symbols:

T	true
F	false

Lastly, any arithmetic expression may be used in place of a parameter name, and complicated blocks involving several names and operators are also permitted. Although in this latter case, it is worth noting that the language capability far surpasses any requirements experienced to date in formulating operating decision systems.

In writing structure tables, the situation often arises where, except for one or two special situations, one course of action is adequate for all input values. The concept of an "all other" row was introduced to avoid enumerating all possible logical combinations of the decision parameter values. The "all other" concept can be verbalized as follows: "if no solution has been found in the table thus far, the solution is in this last row regardless of the problem values." While this greatly reduces table size, it also implies that the problem was stated correctly and does indeed lie within the boundaries of the decision system. The related concept of "all" which appears in the Transportation: New York-Boston, p.m. can be similarly verbalized: "regardless of the problem value proceed to the next column." It was introduced so that a given table need not contain all permissible states of any given decision parameter and also to handle the case where a test in a given column had no significance. In all the above situations the appropriate structure table blocks are left blank signifying no test.

Structure Table Results

Similarly structure table results are not limited to assigning alphabetic constants or numeric values to the result functions or actions named in column headings to the right of the double line. Actually there are four result functions:

"ASSIGN" - which is implied when a named field appears as a result function. This indicates that the result value appearing in (or named by) the solution row is to be assigned or placed in the field named in the column heading.

"CALCULATE" - which is implied by the use of an equal sign after a name appearing as a result value. This indicates that the results of the formula evaluation named in the structure table block should be assigned to the field named as the result function in the column heading. Actually this is not the only way to perform calculations as any arithmetic expression may be used as a result value.

PERFORM - which performs the data processing or arithmetic operations referred to in the label appearing in the result value block. When this is completed, the next result function is executed.

GO - links the structure table to the label appearing in the result value block. There is no implied return in a GO function.

Most of these result functions are illustrated in Figure 3 and Figure 4. In Figure 4, for example, TABLE 2000 assigns the alphabetic constant "FLAT-STRIP" to ASSEMBLE. In the first and third result columns, arithmetic expressions appear as result values. In TABLE 2005 the implied CALCULATE is used for formula evaluation. TABLE 2005 also uses the PERFORM function to solve TABLE 2008 or carry out some other data processing operations depending on the particular solution row. TABLE 2005 is linked by the GO operation to TABLE 2010, 2015, 2020.

TABLE 1005 in Figure 3 shows an interesting use of the GO function. After the winding has been specified in TABLE 1000, assumedly on a lowest cost basis, the product engineer evidently wants to check the insulation temperature rating with the maximum expected operating temperature. If the insulation temperature rating should turn out to be greater everything is fine and the decision-maker proceeds to TABLE 1007. If not, first TYPE-N and then TYPE-T insulation are specified to

supercede TYPE-F, thus getting progressively higher insulation temperature ratings by redirecting the structure table to solve itself.

Frequently, a result function or action will not have a value for all rows. This is common when several result functions are determined by the same structure table. In this situation the phrase "not exist" has been used in verbalizing and the structure table block is left blank.

The use of formulas as structure table results can greatly reduce the size of the table. As an illustration, suppose that a given result function has twenty-six values (10, 12, 14 16, ...60). Ostensibly, the structure table to select the appropriate result value would have twenty-six rows. This decision could be reduced to one row by calculating the result value as some function of the decision parameter as shown in Figure 6. Obviously, all result relationships are not so conveniently proportional but a surprising number of result functions can be described with simple linear and exponential expressions. The curve fitting problem can be greatly simplified by using structure table rows to break the curve into convenient intervals that can be represented by such simple mathematical expressions.

Preambles and Postscripts

Each structure table is preceded by a heading which identifies the table by number and indicates its dimensions in terms of decision parameter columns, result function or action columns, and value rows. Tables may be numbered from TABLE 1 to TABLE 9999999 and allowance is made for up to 999 decision parameter or result functions. Provision is also made for 999 condition rows.

Following the heading is a NOTE which may contain any combination of alphabetic or numeric characters. The NOTE may be used to give the structure table an English name and provide a verbal description of the decision being made. Subsequent to this any labels naming expressions or arithmetic calculations referred to by "CALCULATE" or PERFORM operators in the body of the structure table may be defined. For example, note the definition of TIME ~ 1 and TIME ~ 2 in TABLE 2005 of Figure 4. The structure table proper follows BEGIN.

If no solution row is found in the structure table proper, or if the structure table has executed all results or taken all actions without reaching a GO function then control is passed to the area directly below the structure table. Here are recorded any special instructions pertaining to that particular decision. Of particular note is the situation where no solution row has been found. Such a failure is regarded as an "error." In certain types of decision systems, this may be exactly what the systems designer intended. However, error conditions most often indicate a failure of the decision logic to cope with a certain combination of input values. The systems designer should set up to notify himself whenever such an error occurs by designing an error routine which will provide him with a source language printout identifying the table that failed and the problem being solved at the time. With this problem printout and the source language structure tables, the systems designer has all the data he needs to trouble shoot the system in his own terminology. Thus, each structure table should be followed by the statement: IF NOT SOLVED GO_____. In this way any structure table failures will always be uncovered. Frequently, the situation arises, as mentioned earlier, that regardless of the solution row, the next structure table solved is the same. In this case the statement: GO_____. may be written after or below the preceding error statement, to serve as a universal link to the next structure table.

The areas immediately preceding and succeeding the structure table proper may also be used for input-output, data movement, and other data processing operations.

The Dictionary

The precise name and definition of each decision parameter and result function are recorded in a "dictionary." This dictionary becomes an important planning document in the systems engineer's work for it provides the basic vocabulary for communicating throughout the entire decision system. The dictionary should note a parameter's minimum and maximum values, as well as describe how it behaves. If the parameter is non-numeric in nature, the dictionary should record and define its permissible states. Significantly, the systems engineer formulates both the structure table and the dictionary using his own professional terminology.

The dictionary will also prove useful in compiling and editing structure tables for computer solution. It also follows that problems presented to the resulting operating decision system must also be stated in precisely the same terms as the structure tables. To those as yet uninitiated to the perversity of computers, this may seem a simple matter; unfortunately,

it is not. Interestingly however, one of the more promising application areas for structure tables appears to be in stating the logic for compilers and edit programs.

Summary

The foregoing description of decision structure tables is not meant to be a fully definitive language specification. The intention is to introduce the reader to the decision structure table concept and to discuss their characteristics in sufficient detail to provide the reader with enough understanding to evaluate their inherent flexibility and application potential. Many additional features are available which aid in formulating concise, complete decision structure table systems and also to facilitate input-output operations, but the reader will find that the fundamentals already described are adequate for structuring most operating decision logic.

Automatic Solution of Structure Table Systems

Decision structure tables have proven to be an excellent method for analyzing or formulating the logic of complex industrial information systems, but after taking such great care to precisely record each elementary decision in this highly structured format, it is only natural to speculate on the possibility of solving structure tables automatically with an electronic computer. Before plunging into the computer world, however, it is worth noting that some systems engineers have had very favorable experience using structure tables on a manual basis -- especially as a problem analysis technique, and also in limited applications in manual clerical systems.

Numerous methods for solving structure tables automatically suggest themselves. First, the tables could be coded by hand. Such an approach would use structure tables as a direct substitute for flow charts. Actually this really isn't as bad as it initially sounds. Many benefits would accrue from making this precise readable format the standard method for stating decision logic. It also offers the possibility that a series of macro-instructions could be developed, thereby permitting untrained personnel to code tables without detailed knowledge of computers or programming. However, this approach suffers some distinct disadvantages in comparison with the other alternatives outlined below.

Second, a generalized interpretive program could be written to solve any structure table. This offers the possibility of using a translator to work directly from keypunched structure tables without any manual detail coding. This approach makes economical use of memory since the basic programming to solve any table appears only once and the structure table itself offers a compact statement of decision logic. This reduces the amount of reading time required to bring the problem logic into the computer. File maintenance via recompiling structure table tapes also appears quick and simple. However, interpretive programs usually run more slowly; and this implies some penalty in total machine running time.

A third approach would be to create a structure table program generator in which an object computer program would be generated from the source structure tables. This approach would provide faster computer running times for maximum efficiency. A generator program would probably require more complicated coding than an interpretive translator. In addition, the generated object program would not be as concise as the structure tables themselves. However, where computer running time is of paramount concern, this approach has considerable appeal.

Because of the available time and money, all the early efforts of the Integrated Systems Project toward automatic structure table solution were essentially interpretive. It is interesting that a simple, yet adequate, tabular systems-oriented language could be provided in this way for somewhat less than a man year's effort. Similarly work to date in the area of formula calculations indicates that a comprehensive system of mathematical notation like that required for scientific work is probably not necessary in many operating business decision systems. Initial efforts on the IBM 702 were followed with experimental TABSOL languages for the IBM 305, IBM 650 and the IBM 704. These applications to different computers represented more than simple extrapolations to different pieces of hardware. In each an effort was made to expand capabilities of the language. In addition, the peculiarities of the equipment were explored, since one great concern was to free the user from a programming system usable on one and only one computer. As you might suspect, this wasn't always completely possible on the smaller computers, lacking tape or core memories. Nevertheless, the most recent Manufacturing Service effort on the IBM 650 produced a language with named fields, indexing, a two-address arithmetic, completely generalized structure table formats, and considering the alphabetic restrictions of the

equipment, remarkably flexible output formats.

Although these experimental languages proved quite adequate, one could not help but look toward the tremendous power of one of the more conventional languages. For one thing, the prospects for structure table application in other problem areas brightened, and it seemed reasonable that this power would be desirable in future work. Further our own tabular systems language development had brought us to the point of direct competition with the major language efforts already underway. Here General Electric's Computer Department entered on the scene. The Computer Department was developing a new concept in compiler building for use with General Electric computers. The first version of this new General Compiler, called GECOM, will be available to GE 225 users in May, 1961. It is designed primarily around COBOL, with some of the basic elements of ALGOL, and is now to contain all of TABSOL. To state the results of joining TABSOL with GECOM simply, it places the power of a full-fledged language at the command of every structure table block. Within General Electric, we obviously have a very high regard for the contribution of decision structure tables in information systems design. Significantly, the same committees who developed COBOL are now actively investigating tabular systems-oriented languages as the language of the future. By drawing on the CODASYL work and utilizing the extensive research and development experience already available within General Electric, the Computer Department expects that GECOM will provide users with a system compatible with both the present-day common business language, COBOL, and also the tabular systems-oriented language, TABSOL. Incidentally, the decision structure tables appearing in Figures 3, 4 and 5 are written in conformance with GECOM specifications.

Applications of Structure Tables

As somewhat implied in the illustrations a substantial amount of experience has been gained in applying structure tables to a wide variety of operating decision-making problems over the past three years. But perhaps the most interesting experience, at least from the researcher's point of view, was the very research work which spawned decision structure tables themselves. Earlier, it was mentioned that the Integrated Systems Project undertook a careful study of the essential information and material processing required to directly transform customer orders into finished products. For example, the product must be engineered prior to shipment, but the payroll, though revered by all of us can well be done at some other time, out of the main flow of events. Using this rough rule of thumb, the following activities were studied (Figure 7): order editing, product engineering, drafting, manufacturing methods, and time standards, quality control, cost accounting, and production control. These activities account for a fairly substantial portion of the business system. Normally, they would include 100% of the direct labor and 100% of the direct material as well as about 50% of the overhead. All the production inventory investment lies within the scope of this system and obviously most of the plant and equipment investment. Fortunately, the inputs and outputs to this system are simple and well-defined: the customer order comes in and the finished product goes out. With this in mind, it was possible to treat all activities within these bounds as one integrated, goal-oriented operating decision system and develop decision structure tables accordingly. Working with a small product section in one of the Company's Operating Components, a significant portion of the functional decision logic was successfully structured. Further the resulting structure tables were directly incorporated into a computer-automated operating decision system which transformed customer orders for a wide variety of finished products directly into factory operator instructions and punched paper tape to instruct a numerically programmed machine tool. This prototype system was demonstrated to General Electric management in November, 1958. Starting at the beginning, (Figure 8) the computer system edited the customer order and using the product engineer's design structure tables, developed the product's component characteristics and dimensional details. These in turn were used in the manufacturing engineer's operation structure tables to develop manufacturing methods and determine time standards. And so the flow of information cascaded down through the various business functions computing the quality control procedures, the product costs and the manufacturing schedules; eventually issuing shop paperwork and machine program tapes.

Since the completion of this work further research and development of the structure table concept was conducted in a variety of functional areas for different kinds of businesses in General Electric: defense, industrial apparatus, and consumer-type products. In addition, structure tables have been used in entirely different applications such as compilers. They also appear to hold great promise in complex computer simulation programs.

Benefits of Structure Tables

As a result of these efforts, we have come to believe that the decision structure table is a fundamental language concept which is broadly applicable to many classes of information processing and decision-making problems. They offer many benefits in learning, analyzing, formulating and recording the decision logic:

1. Structure tables force a logical, step-by-step analysis of the decision. First the parameters affecting the decision must be specified; then suitable results must be formulated. The nature of the structure table array is such that it forces consideration of all logical alternatives, even though all need not appear in the final table. Similarly, the precise structure table format highlights illogical statements. This simplifies manual checking of decision logic. The decision logic emphasizes causal relationships and constantly directs attention to the reasons why results are different. Personal design preferences can be resolved and intelligent standardization can be fostered.

   This is no mean capability. Indeed, it was very instructive to witness the development of methods and time standards logic in parallel with the development of the engineering logic during the initial Integrated Systems Project study. Through analysis of the decision structure tables written by the various functional specialists, everyone was able to achieve an insight into the product and the business rarely obtained in so short a period of time. The facts of life in product design, factory methods, and standardization were brought into the open very rapidly.

2. Structure tables are easily understood by humans regardless of their functional background. This does not imply that anyone can design or create new structure tables to describe a particular decision-making activity; but it does mean that the average person, with the aid of a dictionary, can readily understand someone else's structure tables. Thus, structure tables form an excellent basis for communication between functional specialists and systems engineers. Structure tables also go a long way toward solving the difficult systems documentation problem.

3. Structure table format is so simple and straightforward that engineers, planners, and other functional specialists can write structure tables for their own decision-making problems with very little training and practically no knowledge of computers or programming. Given a few ground rules, regarding formats and dictionaries, the structure tables written by these functional people can be keypunched and used directly in operating decision systems without ever being seen by a computer programmer. This cuts computer application costs as well as cycle times.

4. Structure table errors are reported at the source language level, thus permitting the functional specialist to debug without a knowledge of computer coding.

5. Structure tables solved automatically in an electronic computer offer levels of accuracy unequalled in manual systems. Note, however, that any such mechanistic systems lose that tremendous ability of humans to compensate for errors or discrepancies.

6. Structure tables are easy to maintain. Instead of changing all the precalculated answers in all the files, it is often only necessary to change a single value in a single table. For example, when changing the material specified for a component part under current file reference systems, it would be necessary to extract, modify and refile all drawings and parts lists calling for any variation of the component part. Using structure tables, it would only be necessary to alter those structure tables which specified the component material.

Summary

In closing, we recommend that the reader demonstrate the effectiveness of decision

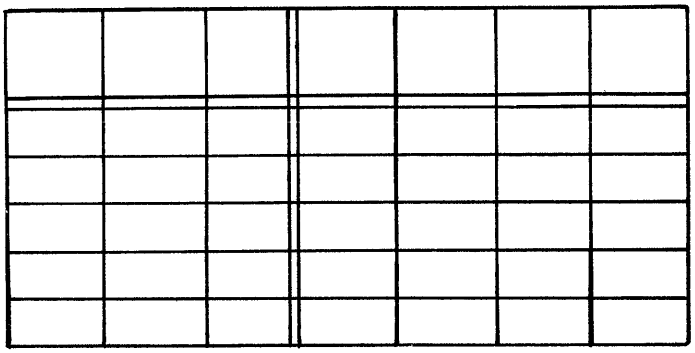
structure tables to himself by "structuring" a few simple decisions. For example, write a structure table which will enable your wife to decide how to pack your suitcase of any business trip. Perhaps a simple business decision such as those mentioned earlier would provide a more instructive example. The first structure tables are usually difficult to write, because most of us do not, as a general rule, probe deeply into the logic supporting our decisions. However, once this mental obstacle is overcome, "structuring" facility develops rapidly. If the reader will take the time to "structure" a few decisions and actually experience the deeper insight and clarity which this technique provides, then decision structure tables need no apologist, they will speak for themselves.

Acknowledgement

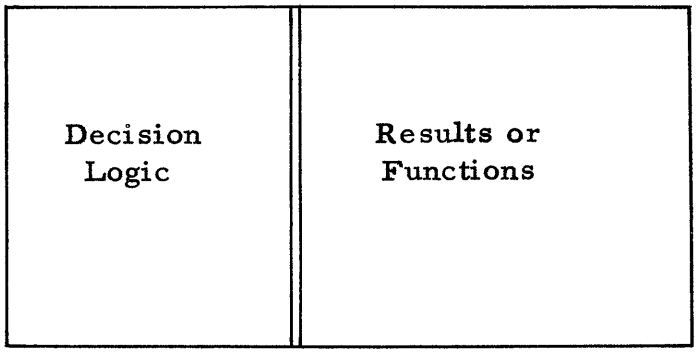
In contrast to most technical papers which essentially document only the work of the author, this discussion reports on the efforts of over seventy-five General Electric men and women. In particular, credit is due Mr. Burton Grad, who though no longer with General Electric, was a principal originator of the decision structure table concept. Mr. Malcolm C. Boggs, Mr. Daniel F. Langenwalter, Mr. Herbert W. Nidenberg, and Mr. Theodore E. Schultz representing Service Components and personnel from some fifteen different Operating Components within General Electric have contributed toward bringing these ideas to their present state of development and application. Acknowledgement is also due Mr. Charles Katz of General Electric's Computer Department who was instrumental in joining TABSOL and GECOM.

Decision Structure Table

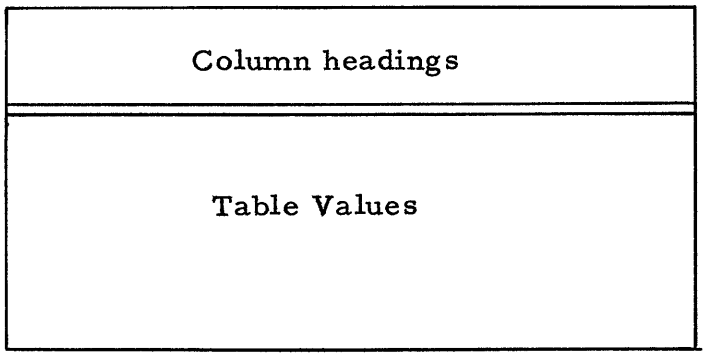
...a rectangular array of terms, or blocks...



...vertical double line...



...horizontal double line...



...structure table values...

P_{01}	P_{02}	P_{03}	R_{01}	R_{02}	R_{03}	R_{04}
p_{11}	p_{12}	p_{13}	r_{11}	r_{12}	r_{13}	r_{14}
p_{21}	p_{22}	p_{23}	r_{21}	r_{22}	r_{23}	r_{24}
p_{31}	p_{32}	p_{33}	r_{31}	r_{32}	r_{33}	r_{34}
p_{41}	p_{42}	p_{43}	r_{41}	r_{42}	r_{43}	r_{44}

Figure 1

Problem Statement: Select Transportation, New York - Boston, p.m.

Weather: Foul

Plane Space: OK

Hotel Room: Open

Decision Structure Table: Transportation, New York - Boston, p.m.

Weather	Plane Space	Hotel Room	Trans-portation	Other In-structions	Next Decision
Fair	OK	Open	Plane		End
Foul	OK	Open	Train	Cancel Plane	End
	Sorry	Open	Train		End
	OK	Filled		Cancel Plane	NY-Bost. a.m.
	Sorry	Filled			NY-Bost. a.m.

Solution:

If the value of Weather is Foul, and

the value of Plane Space is OK, and

the value of Hotel Room is Open,

Then

the value of Transportation is Train, and

the value of Other Instructions is Cancel Plane, and

the value of Next Decision is End.

Figure 2

TABLE 1000. DIMENSION C4 A5 R10.
NOTE TABLE FOR DETERMINING DETAIL VARIABLE PART CHARACTERISTICS FOR A LINE OF SENSING COILS IN ACCORDANCE WITH CUSTOMER END PRODUCT SPECIFICATIONS.
BEGIN.

SERVICE EQ	UNITS EQ	VALUE	VALUE	TURNS	DIA	RESIST	INSUL	INSUL ~TEMP
"DC"	"MAMP"	GR 180	LS 450	0.3/I	.001	2.6*TURNS	"TYPE-F"	150
:	:	:	:	:	:	:	:	:
"DC"	"MVLT"	GREQ 45	LSEQ 150	26	.008	1.84	"TYPE-F"	150
"DC"	"MVLT"	GR 150	LSEQ 330	13	.002	0.46	"TYPE-F"	150
"DC"	"VOLT"	GREQ 0.9	LSEQ 300	60	.002	39.0	"TYPE-F"	150
"DC"	"VOLT"	GR 300	LSEQ 1100	120	.002	137.0	"TYPE-F"	150
:	:	:	:	:	:	:	:	:
"AC"	"WATT"			230	.002	150.0	"TYPE-N"	200

IF NOT SOLVED GO ERROR~COIL.
MOVE "COPPER" TO MATERIAL.
GO TABLE 1005.
END TABLE 1000.

TABLE 1005. DIMENSION C2 A3 R3.
NOTE TABLE TO MAKE CERTAIN THAT INSULATION TEMPERATURE RATING EXCEEDS MAXIMUM OPERATING TEMPERATURE.
BEGIN.

MAX~TEMP	INSUL	INSUL	INSUL~TEMP	GO
LSEQ INSUL~TEMP				TABLE 1007
GR INSUL~TEMP	"TYPE-F"	"TYPE-N"	200	TABLE 1005
GR INSUL~TEMP	"TYPE-N"	"TYPE-T"	250	TABLE 1005

IF NOT SOLVED GO ERROR~COIL.
END TABLE 1005.

Figure 3

TABLE 2000. DIMENSION C3 A3 R4.
NOTE TABLE TO SPECIFY VARIABLE FACTORY OPERATION CHARACTERISTICS FOR THE INITIAL SENSING COIL WINDING FROM PART CHARACTERISTICS.
BEGIN.

SUPPORT~TYPE EQ	MATERIAL EQ	TURNS	START~W	ASSEMBLE	FINISH~W
"TABED-HOLE"	"COPPER"		TURNS		
"FLAT-STRIP"	"COPPER"	LS 100	2	"FLAT-STRIP"	TURNS-2
"FLAT-STRIP"	"COPPER"	GREQ 100	TURNS/2	"FLAT-STRIP"	TURNS/2
"FLAT-STRIP"	"ALUMNM"		TURNS	"2 FLT-STRP"	

IF NOT SOLVED GO ERROR~COIL. GO TABLE 2005.
END TABLE 2000.

TABLE 2005. DIMENSION C2 A3 R3.
NOTE TABLE TO CALCULATE TIME STANDARD FOR PREVIOUS OPERATION.
TIME~1 = 125*DIA*TURNS.
TIME~2 = 1000*DIA/SQRT (TURNS).
BEGIN

TURNS	TURNS	TIME	PERFORM	GO
LS 15		TURNS + 0.88	SETUP	TABLE 2010
GREQ 15	LS 100	TIME~1 =	SETUP	TABLE 2015
GREQ 100		TIME~2 =	TABLE 2008	TABLE 2020

IF NOT SOLVED GO ERROR~COIL.
GO TABLE 2005.

Figure 4

TABLE 1010. DIMENSION C2 A1 R3.
NOTE COIL QUANTITY DETERMINATION.
BEGIN.

SERVICE EQ	UNITS NEQ "WATTS"	COIL~QUAN
"AC"		0
"DC" OR "AC"	T	QUAN
"DC"	F	2*QUAN

IF NOT SOLVED GO ERROR~COIL. GO TABLE 1100.
END TABLE 1010.

TABLE 1500. DIMENSION C4 A3 R10.
NOTE COIL LOAD DATA AND CYCLE TIMES.
BEGIN.

SERVICE EQ	UNIT EQ	ACY EQ	INSP EQ	NORM~CYCLE	MIN~CYCLE	COIL~LOAD
"AC"	"AMPS" OR "MAMP"	1	"COML"	15	11	QUAN
"AC"	"WATT"	1	"COML"	15	11	2.2* QUAN
:	:	:	:	:	:	:
"DC"	"AMPS" OR "MAMP"	2	"COML"	15	9	0.9* QUAN
"DC"	"VOLT" OR "MVLT"	2	"COML"	15	9	0.9* QUAN
"DC"	"AMPS" OR "MAMP"	1	"GOVT"	20	16	1.4* QUAN

IF NOT SOLVED GO ERROR~COIL.
MIN~DATE = TODAY + MIN~CYCLE.
NORM~DATE = TODAY + NORM~CYCLE.
GO TABLE 1510.
END TABLE 1500.

Figure 5

TABLE 1510. DIMENSION C2 A2 R3.
NOTE COIL PROMISE DATE DETERMINATION.
BEGIN.

COIL~LOAD LSEQ	CUST DATE	PROMISE	GO
CUM~CAP (NORM~DATE)	GREQ NORM~DATE	CUST~DATE	NORM~LOAD
CUM~CAP (MIN~DATE)	GREQ MIN~DATE	CUST~DATE	RUSH~LOAD
CUM~CAP (CUST~DATE)		CUST~DATE	EMER~LOAD

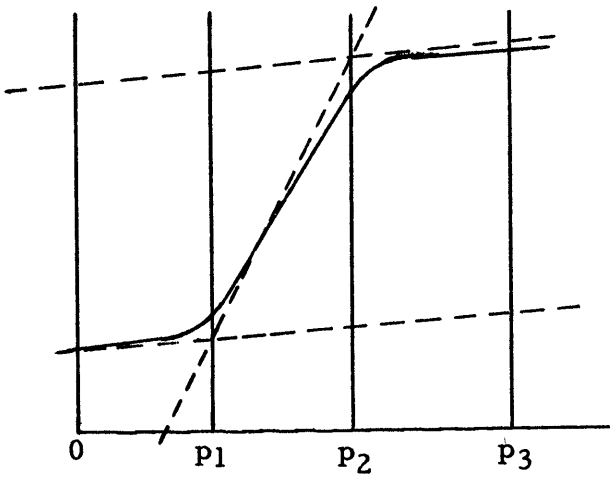
IF NOT SOLVED GO OVERLOAD.
END TABLE 1510.

Figure 5a

P	R
0	10
1	12
2	14
3	16
.	..
.	..
.	..
.	..
25	60

P	P	R
0	25	(2*p) + 10

.... The use of formulas as structure table results can greatly reduce structure table size, as shown by the simple straight line expression above. Structure tables may also be used to partition complicated curves into convenient segments as shown below. ....



P	P	R
0	p_1	lx + a
p_1	p_2	mx + b
p_2	p_3	nx + c

Figure 6

PRESENT MAIN LINE SYSTEM

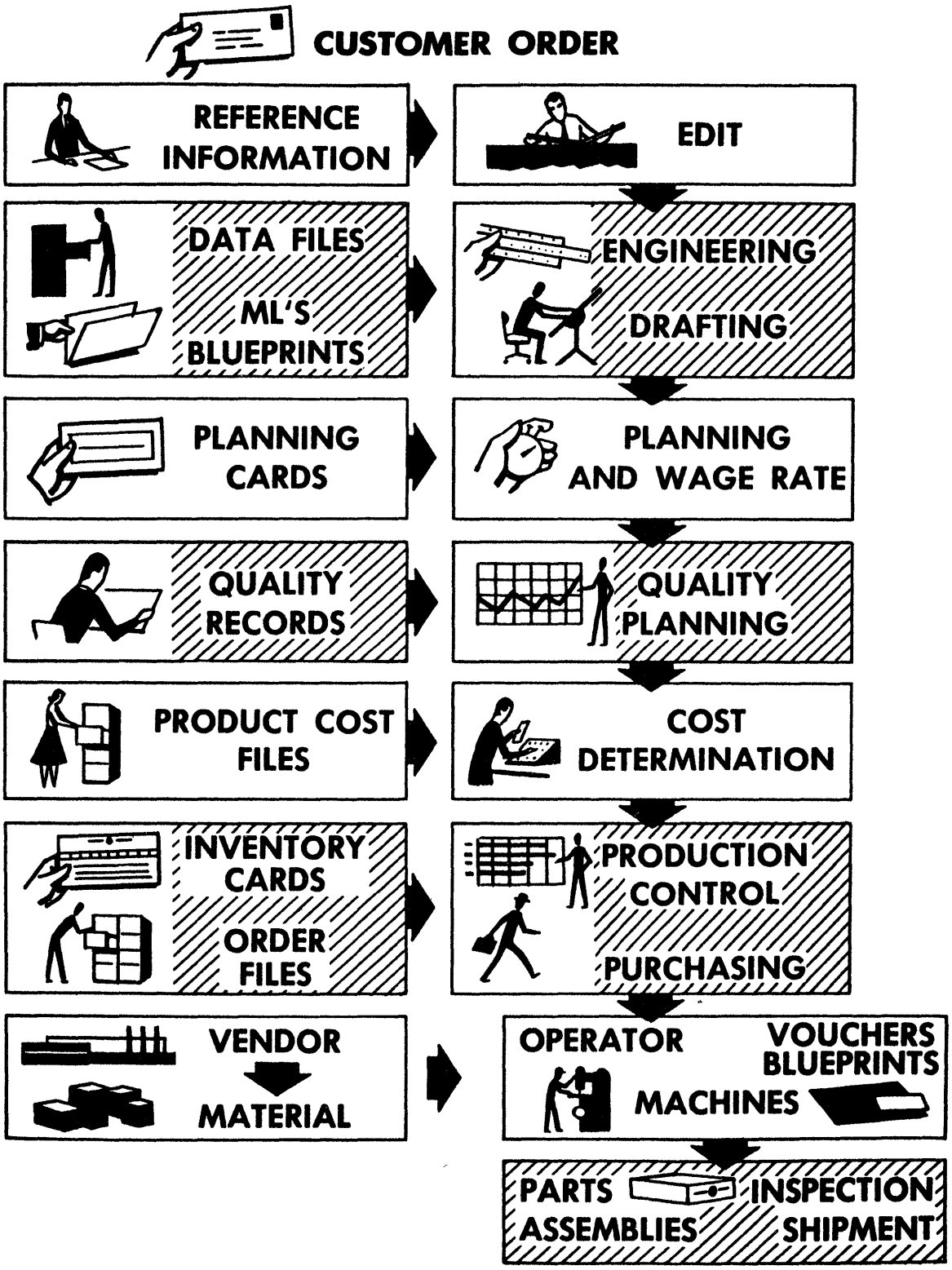


Figure 7

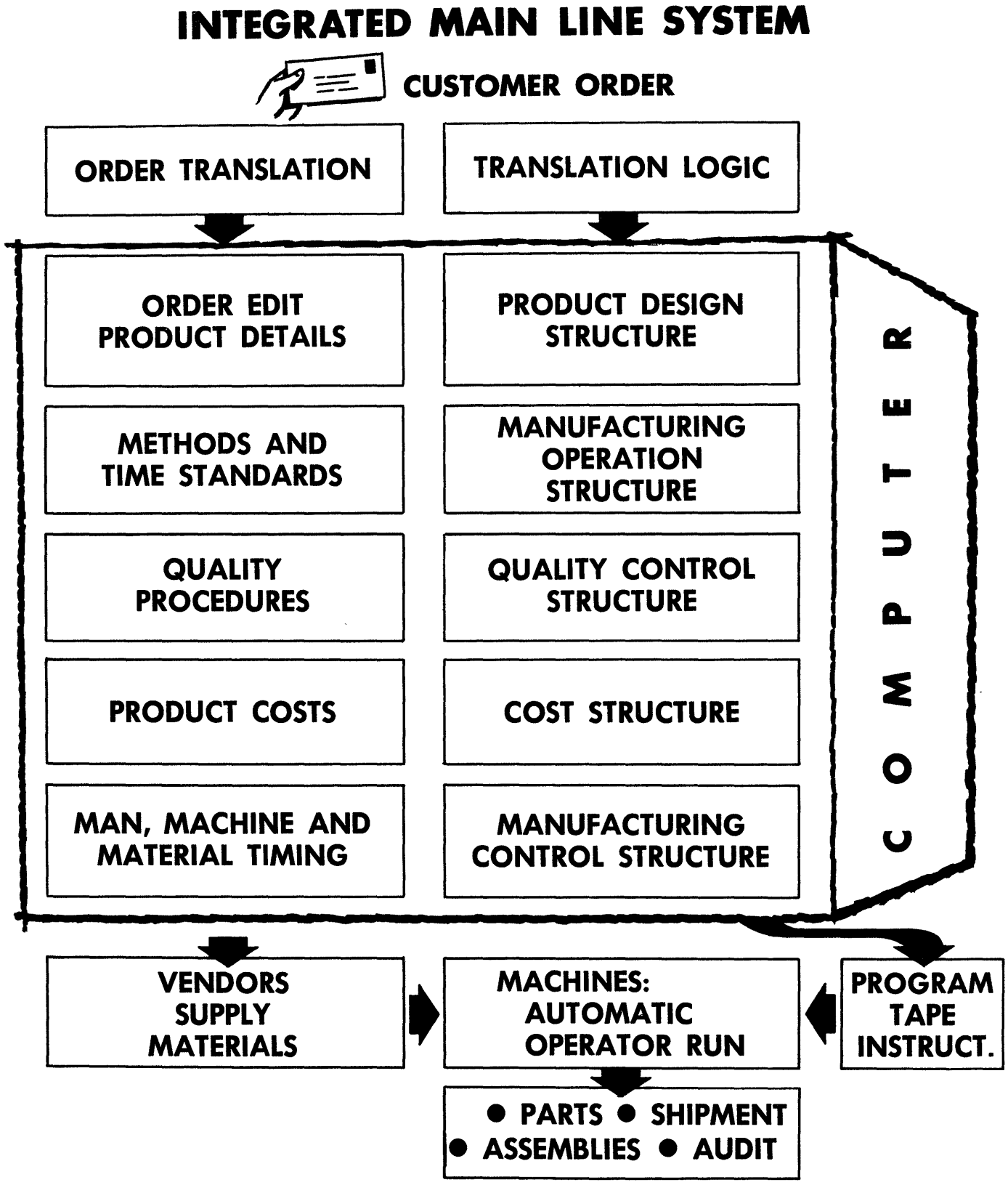

INTEGRATED MAIN LINE SYSTEM
CUSTOMER ORDER
ORDER TRANSLATION
TRANSLATION LOGIC
ORDER EDIT
PRODUCT DETAILS
PRODUCT DESIGN
STRUCTURE
METHODS AND
TIME STANDARDS
MANUFACTURING
OPERATION
STRUCTURE
QUALITY
PROCEDURES
QUALITY CONTROL
STRUCTURE
PRODUCT COSTS
COST STRUCTURE
MAN, MACHINE AND
MATERIAL TIMING
MANUFACTURING
CONTROL STRUCTURE
COMPUTER
VENDORS
SUPPLY
MATERIALS
MACHINES:
AUTOMATIC
OPERATOR RUN
PROGRAM
TAPE
INSTRUCT.
● PARTS ● SHIPMENT
● ASSEMBLIES ● AUDIT

Figure 8

THEORY OF FILES *

Lionello Lombardi

University of California at Los Angeles

Summary

The theory of files is a tool for the logico-mathematical treatment of automatic non-numerical data processing problems, such as machine accounting, information retrieval and mechanical translation of languages. The main result which has been obtained sofar from the application of the theory of files is the formulation of a simple pattern to which the data flow of any information processing procedure conforms, regardless of how many files are involved. The flow of each file can be controlled and coordinated with the flow of the other files by means of five boolean parameters, called 'indicators'.

A specially designed Algebraic Business Language exploits this result for the purpose of programming digital data processing systems. This paper also probes into the impact of the theory of files upon the logical design of digital information processing systems.

Introduction

The first step of any initiative yielding to the scientific knowledge of a new field consists of the definition and development of a language and of a notation able to describe the phenomena which characterize such a field. In particular, the adaptation and the adoption of one specific language - the mathematical language - has been successful in several areas where the need for a scientific investigation existed.

That aspect of human activity which seems to be growing the fastest (in such a way that it sometimes threatens to minimize the importance of all the others) is the control of paperwork. Paperwork is one of the most impressive products of civilized society; its relevance with respect to the other activities swells with the progress of our economy and technology in a way which is liable to jeopardize this progress itself. Paperwork is generally carried out by machines; however, the work of organizing, coordinating, defining and describing it is still performed by humans. The proportion of the total available manpower that it absorbs grows dangerously with the wealth and sophistication of our society. The phenomenon of paperwork control has reached the stage where it should be investigated scientifically, hopefully to repeat the success that comparable scientific approaches yielded when they were applied to other fields, in terms of promoting the knowledge, suggesting the development of suitable techniques, allowing the adoption of reliable procedures and collapsing the amount of work necessary for carrying them out. We believe that this can be done, and that the appropriate language for analyzing coordinated paperwork can only be mathematics. The purpose of the theory of files is to support this belief.

Today the theory of files, whose basic definitions are stated in[1], has been applied only to investigation of a specific area of paperwork control: the area of systems analysis, namely of the analysis and definition of those procedures which can be carried out by automatic data processing systems. The specific problem of non-numerical data processing has been emphasized, the main reason for this being the widely spread prejudice that this field cannot possibly be approached scientifically.

Available Computer Languages

A wide selection of computer languages designed with the aim of providing tools for automatizing the programmer's work is available today; however, it seems that none of these languages can help the systems analyst. We think that the main shortcoming of such languages (which range from simple assemblers to autocoders able to handle macros, and to such languages as the IBM Commercial Translator, COBOL, FACT, Flow-Matic or AIMACO), are all designed according to the pattern that we called 'v. Neumann Language' in[2]: a v. Neumann language[4] is a language in which a phenomenon is described by means of a sequence of statements divided in two categories - 'executable' statements and 'descriptive' statements - in such a way that the statements of the first category can be put into a one-to-one correspondence with a flowchart[5] of the procedure involved by the phenomenon represented.

Such a language can be used successfully for describing layouts of information supports and sequences of actions, namely procedures. Unfortunately such languages cannot possibly provide for a synthetical and compact definition of the compound of logical conditions to which any action is subjected, nor for the synthesis of a coordinated flow of information. For instance, if we consider a language such as COBOL and we try to use it for representing integrated data processing procedures, the following shortcomings come to light:

(1) Each statement has the form "IF condition THEN action", where the action denotes a sequence of steps and the condition denotes a boolean expression. Nevertheless the execution of the action is only apparently fully controlled by the condition, in the sense that the value 'true' of the condition is necessary but not sufficient for the execution of the action. Such execution depends also upon the path of the control through the procedure description, i.e, on the result of several tests, some of which may have preceded by far the action in question. Since the possible path of the control are myriad, then, for determining the circumstances under which a certain action is to be executed, one should carefully trace through all of the procedure description. It is usually not practically feasible even to identify such circumstances, nor to correlate an action to the original input information. What is needed is a language by which all of the conditions affecting an action are compounded into a unique statement: such a language could not possibly contain any control statements, and consequently could not possibly be a v. Neumann language.

(2) Most documents are self-explaining, as far as their path between different procedures is concerned. However, one of the big problems in systems analysis is the determination of when and under which circumstances a document is entered into or issued from a procedure. This well known problem of efficiently coordinating the data flow becomes one of the main issues of systems analysis whenever one wants to save computer time by increasing the degree of procedural parallelism. In COBOL, like in any other v. Neumann language, the flow of information can only be controlled by means of input-output statements, and by a proper organization of the control statements, i.e, by carefully planning the flow of the machine control through the procedure description. In such easy applications as payroll, where the degree of parallelism cannot possibly be high, COBOL can be used successfully. On the contrary, consider applications in which, for the sake of saving computer time, several functionally independent procedures that relate only by the fact that they operate on sets of files which are sorted with respect to the same key (i.e., they are equiordered), are run in parallel: then the use of such a language leads to long and exceedingly involved descriptions. Even in some comparatively simple applications, where for example, billing and accounts receivable (or ordering and accounts payable) are run in parallel, together with the updating of a master file and with the preparation of data to be used later by the management (such as notes for exceptional cases, requested reports, or totals), COBOL can compare favorably to some of the available autocoders as a programming language. However, it does not seem to be an appropriate analysis language for such applications. In cases where the degree of parallelism is high and the data flow is complex, such a language should be discarded.

(3) Last and least, it appears that the use of some type of kindergarten English, whose adoption seems to be due to the objectionable assumption that it is more readily understood by top executives than any more appropriate technical notation, is an obstacle to the use of COBOL even as a programming language, because it yields comparatively long procedure descriptions. However, this last shortcoming is really irrelevant, primarily because it affects COBOL only as a programming language. Further this shortcoming can be removed easily from the language without any major change in the logic of its translators. It has also been a common experience of individuals programming with COBOL that after a few statements one drops the English of the language and uses abbreviations, especially for such phrases as "IS GREATER THAN".

Basic Ideas

In addition to the trend which finally led to COBOL, two independent ideas were developed in the past few years. Both aimed at the creation of a system language suitable for describing non-arithmetic data processing procedures. The first philosophy can be summarized as follows: 'We must algebrize the non-numerical procedures, in order to be able to apply to them successful algebraic languages such as Fortran[6]'. The other can be expressed in this way: 'The major problem in non-arithmetic data processing is the one of defining and coordinating the data flow: before we can design a system language, we should discover and formulate the laws of the data flow'.

The theory of files is but a synthesis of these two ideas: from a methodological standpoint the theory of files consists of expressing and analyzing algebraically the laws of the data flow.

The starting point in the theory of files was the remark that if we consider the merger between two files of records as a sum, while the merger of them with selection of all those records whose key is not present at least once in both files as a product, then, under certain circumstances, sets of equiordered files can be reduced to boolean algebrae of files. In such an algebra the most common file handling operations can be defined by simple algorithms: for example, a k-way sorting-by-merging procedure (either fixed or variable length-sequence) is represented as a recurrent summation of k files.

From a file-theoretical standpoint, a procedure is broken down into a sequence of PULSES, at whose beginning new records are (logically) entered, during which calculations are performed, and at whose end all the completely processed records are (logically) filed. Only records whose keys all have the same value are considered in a pulse. A sequence of pulses during which all the records of all the files involved in a procedure, whose keys equal a certain constant, are processed, is called a PHASE of the procedure.

The language that we propose for describing procedures is the Algebraic Business Language (ABL). It is described in[1], where the basic concepts of the theory of files are defined mathematically. In its simplest version, an ABL procedure description consists of a sequence of 'conditional expressions', namely of sets of executive orders ('actions') subject to boolean expressions ('conditions'). There are no control statements, and the conditional expressions are to be considered sequentially[8], i.e., from the first to the last.

Optimization.

Simplification. In each procedure, a LOGICAL ORDER of the files involved must be given by the analyst.

Definition 1: "Let us denote by DD the data description of a given problem; then two procedures are 'DD - equivalent' if they both transform any input organized according to DD into the same output".

Definition 2: "A procedure P is called 'DD - optimized' if, in the space {DD, P} of all the procedures which are DD - equivalent to P, P both
a) Maximizes the parallelism of logical input-output
b) Minimizes the amount of internal processing".

From an applicative standpoint, only DD - optimized procedures should be considered: notice that the maximization of the parallelism of the physical input-output flow can be obtained only on the basis of a logical one whose parallelism is maximized.

Now two pulses are always independent as far as internal processing is concerned, and two phases are always independent as far as input-output is concerned: consequently, in order to DD - optimize a procedure, it is sufficient to
I) Maximize the parallelism of the logical input-output within the pulse.
II) Minimize the amount of internal processing within the phase.

Since ABL is a sequential language, point II can be accomplished simply by performing a precedence analysis and a simplification of the procedure description. (Notice that this would not be easy in a v. Neumann language).

In order to discuss point I, let us consider separately input-output.

Input. The pattern which maximizes the input-parallelism is unique for any phenomenon of the kind we are considering. More precisely, it consists of the following:
1) No more than one record per each file is entered during any pulse.
2) Consistently with 1), a record belonging to any file F is entered as soon as it is both logically available and all the records pertaining to the current phase, belonging to any one of the files which precede F in the logical order, have been entered.
3) A phase is over at the end of its pulse during which the last record pertaining to it is entered.
4) Logical input is only performed at the beginning of the pulses.

Since the input pattern is unique for any procedure of the type we consider, the analyst is not burdened with the control of the input: he can just forget about it. The only way in which the analyst using ABL can control input is by designing the logical order of the files properly.

Ouput. Unlike input, which is fully standardized and automatic, output is entirely and directly controlled by the analyst. In fact, the conditions under which documents are to be issued always depend upon the particular phenomenon considered. Furthermore, the determination of these conditions can often be considered as the major single factor in the representation of this phenomenon. Since it is important to have these conditions compounded in a single synthetic expression for each output file, the output of each file is controlled by a FLOW CONTROL EXPRESSION, consisting of the name of the file in question followed by a boolean expression denoting the condition under which a record of this file is to be issued.

Indicators. The boolean variables used for writing a procedure description (which are compounded into boolean expressions in ABL, while they consists of sets of parts of different statements in any v. Neumann language) may have three origins:
a) They may be generated by comparison between numeric, alphameric or boolean entities.

b) They may be determinations of conditional variables.
c) They may be references to the current configuration of the data flow.

While no need arises for a special discussion of the conditions of the first two categories, we may point out that when any v. Neumann language representation of a procedure is used, such references are generated by means of careful constructions and comparisons of keys. The theory of files suggests that the layouts of the keys of the files are given as part of the data descriptions, and that the keys of the records are constructed and related automatically to each other as part of the I-ϕ operations: consequently, these operations are not under the control of the analyst. Since references to the current status of the data flow are often necessary for making decisions which condition the phenomenon considered, ABL must have a provision for giving to the analyst complete information about it. The configuration of the data flow never changes during any pulse, and from a file-theoretical standpoint it can be fully characterized by stating the occurrence or omission of five conditions for each one of the files involved. This can be done by means of INDICATORS, five boolean variables per each file, whose value never varies during any pulse.

Let us explain intuitively what each indicator of a file F stands for:
1) the 'EXISTENCE INDICATOR' of F denotes the logical presence of a record of F.
2) The 'LEFT DERIVATIVE' and the 'RIGHT DERIVATIVE' of F characterize those records of F whose key has a value which is different from the value of the keys of all the preceding [following, respectively] records of F.
3) The 'INPUT - OUTPUT INDICATOR' of F by being "on" denotes those pulses where a record of F is entered or issued.
4) The 'NON CONFORMITY INDICATOR' of F characterizesthose records of F which are incomplete or non-conforming.

Four further indicators, which are common to all files, are available in each representation of a phenomenon for denoting its initiation and closure. No other information regarding the data flow is needed in any DD - optimized procedure, in whose description the indicators can be used without any distinction from the other boolean variables.

The setting and resetting of the indicators (i.e., the 'indicator logic') is performed automatically according to the rules stated in[1] (section 3), where the laws of the automatic data flow control-in particular of the input mechanism - are stated in terms of relations between the indicators. The set of values of the indicators in each pulse is a synthesis of the data flow control related to it, and is obtained as a subproduct of the logical operations involved by the input and output.

Though the analyst can neither set nor reset any indicator, an indicator can be used anywhere in the procedure description: in particular in the Flow Control Expressions.

Hardware and Implementation

Sequential Languages

Like a mathematical synthesis of a physical phenomenon can be stated by means of a sequence of equations, so the theory of files allows one to express a mathematical synthesis of a data processing phenomenon in ABL by means of a sequence of conditional expressions . In both cases the sequence is considered from the first equation (or conditional expression, respectively) to the last one. The flow control expressions are conditional expressions where the action consists of issuing a record. Neither the equations of an algorithm nor the conditional expressions of a non-arithmetic procedure description are in a one-to-one correspondence with the steps of any path that a machine control would follow in order to carry them out. Unlike any v. Neumann language, ABL is 'sequential' and asynchronous with respect to the way the procedures described are implemented. Our study shows that languages having this structure are generally more suitable than v. Neumann languages for approaching data processing phenomena scientifically.

If one wants to utilize the theory of files not just as a method of investigation but also as a tool for the automation of systems analysis, he must be able to develop mechanically sequential outlines into flow charts, i.e., into procedures. More precisely, one should be able to transform the sequential representation of any data processing phenomenon into a DD optimized flow chart. Apparently this transformation can quite easily be made because of the standard input scheme, and of the fact that each conditional expression completely determines one specific issue of the procedure, like the presence of a record in an output file or the value of a certain field of an output record, etc. A difficulty arises when we consider the interrelationship between the indicators of the various files; for example, the condition-part of a certain conditional expression, say EA, may depend upon the setting of an indicator of an output file whose records are filed under the control of another Flow Control Expression, say EB, which comes after EA in the sequential description. Consequently, the sequence of operations must be properly arranged in order to avoid unnecessary look-aheads. Such rearrangements should not be performed by the analyst, who should only be concerned with the statement of the information processing effect of the phenomenon, rather than with procedural considerations or with any simplification of

the correlation among expressions. This simplification should be carried out by the machine, together with the entry and removal of auxiliary conditional expressions and with the optimization of the arithmetic formulae. This last operation should not be bounded to the optimization of each single formula within itself, but should consist of analyzing the relations between different expressions in order to avoid unneeded repetitions. The study of Semapraxis codifies the efforts of analysists toward the intelligent utilization of computers for such machine simplifications. In particular[10] by Feldstein enunciates such details.

The ABL representation of a phenomenon can also be mechanically checked against tautologies and contradictions which may depend on an erroneous analysis of the phenomenon itself; for instance, in the above example, if the phenomenon is coherently stated, EB should depend neither directly nor indirectly on EA.

Special Devices.

Most stored program data processors are provided with an operating system which includes efficient buffered input-output subroutines. Some data processors-the IBM 7070-74, for example - have specific features (scatter-read-gather-write, highly parallel memory bus, block transmission with rearrangement, etc.) which allow the programming of very efficient I-$\emptyset$ routines, including the necessary key logic. In accordance with the adoption of some new ideas in the design of machinery, (consider for instance the non-arithmetic processor of the IBM Harvest, or the systems with a Fixed+Variable structure[9]) it is sometimes convenient to wire such routines, which become parts or modules of the hardware.

When a system has to carry out procedures represented in ABL a similar alternative arises for the indicator logic, which will be programmed for standard systems and built for more advanced and specialized ones.

A third case where the issue of a comparison between wired and programmed 'giant commands' varies with the modernity and specialization of design of the basic hardware is related to the handling of the compact and flexible 'table operations' with whose use ABL provides the analyst (see[1], section 2).

The implementation of ABL is significantly conditioned by the hardware considered: it appears more difficult to carry it out for standard, strictly stored-program computers than for more advanced ones. The generation of a program on the basis of an ABL representation is more direct in the last case; we think that this is due to the great deal of overlap among the ideas which led the engineers to such advances in systems design and those which yielded the theory of files.

However, the indicator logic is new only as a method. Most control statements and logical operations written by programmers using COBOL or symbolic machine languages should be considered as a clumsy, approximative and only partially satisfactory replacement for a clean, universal and fully automatic indicator logic.

Let us conclude by pointing out that the advantages of adopting sequential languages does not seem to be bounded to the use of large scale data processing systems. On the contrary, such languages appear to be intimately related to the nature of non-numerical data processing phenomena, regardless of their implementation; for example, a sequential language quite similar to ABL proved to be well suited for representing procedures to be carried out by very simple, externally programmed data processors[3].

*The preparation of this paper was sponsored by the Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the United States Government. The author also wishes to express his gratitude to M. Alan Feldstein for his help in the preparation of this paper.

References

1. L. Lombardi, "Mathematical Structure of Non-Arithmetic Data Processing Procedures" (Forthcoming in J.A.C.M.).

2. L. Lombardi, "System Handling of Functional Operators" (Forthcoming in J.A.C.M.).

3. L. Lombardi, "Inexpensive Punched Card Equipment" (Forthcoming.).

4. H. Goldstine, J. v Neumann, "Planning and Coding for an Electronic Digital Computer" (I.A.S., 1948).

5. IBM Staff, "Flow Charting and Block Diagramming Techniques" (C20-8008).

6. Notice that Fortran is also a "v. Neumann language".

7. See[1], section 4.

8. Another 'sequential language' was conceived independently of us by C.B. Tompkins with the collaboration of M.A. Melkanoff and J.D. Swift.

9. G. Estrin, "Organization of Computer Systems - The Fixed plus Variable Structure Computer" (Proc. of the 1960 WJCC).

10. M.A. Feldstein, "Semapraxis" (Forthcoming).

POLYPHASE MERGE SORTING -- AN ADVANCED TECHNIQUE

R. L. Gilstad
Minneapolis-Honeywell Regulator Company
Electronic Data Processing Division
Wellesley Hills, Massachusetts

The Challenge

Designers of generalized library sort packages for the current and future generations of computers are faced with the challenge of developing new techniques that provide more effective use of these computers. The major concern in developing efficient sorting routines in the past has been the internal sorting techniques, that is, the methods of manipulating the data within the memory of the computer. Precise methods must, of course, be devised for each new computer design but, due to the extensive effort in this area in the past, few new internal sorting techniques have been introduced for, what are in computer terms, generations.

Emphasis is now being given toward more effective use of the tape drives used by a sort routine. Progress toward this end was reported in the paper "New Merge Sorting Techniques", presented by B. K. Betz at the September 1959, ACM Conference. The paper then presented described in theory an advanced merging technique, originally called the "N-1" technique, now a proven method better known as the Cascade sorting technique.

The intention of this paper is to introduce a new merging technique, polyphase sorting. The following section describing the application of the Cascade sorting technique is included to aid in the understanding of the evolution of the polyphase sorting technique and to prepare for certain comparisons later in this paper between the various sorting techniques. A complete study of the changes in merge sorting would, of course, include a description of the process that is referred to in this paper as normal merge sorting and that has such names as two-way merge sorting and three-way merge sorting. Because of the extensive use of normal merge sorting techniques, this paper assumes a general understanding of them by those interested in this subject.

Cascade Sorting

The Cascade Merge Sort, available exclusively in the Honeywell 800 automatic programming packages, is a two-segment program, the first part of which is an internal sort that creates strings of ordered items. The internal sorting method that has proven to be most advantageous to Honeywell for generalized sort generators uses the "tag bin" concept which transfers internally only a tag representing each item stored in memory, instead of transferring the entire item. Further, the "replacement" sorting method is added, which creates strings of ordered items substantially longer than the number of items stored in memory. This method, in fact, provides strings averaging twice the number of items stored in memory for randomly ordered input data and longer strings if any pre-ordering exists in the input file.

The only real difference between the internal sort, hereafter called the pre-sort, for Cascade sorting and normal sorting is the manner of distributing the strings onto the work tapes used by the sort. Normal merge sorts require that the strings be distributed alternately on two work tapes for a two-way merge sort, or on three tapes for a three-way merge sort. The pre-sort for a Cascade sort distributes the strings of sorted records onto all but one of the work tapes available to the sort. The ideal distribution at the completion of the pre-sort is such that there are fewer strings on each succeeding tape. The exact distribution is based on one of several sequences, depending on the number of tape drives being used. The sequence for the three-tape sort is the Fibonacci sequence:

$$1,1,2,3,5,8,13,21.....,$$

while the sequence for a four-tape Cascade sort is the sequence:

$$1,1,1,2,3,5,6,11,14,25,31..... .$$

If the pre-sort for a four-tape Cascade sort creates 14 strings, the distribution of strings on the three work tapes would be six strings, five strings, and three strings.

The determination of the distribution of strings by the pre-sort can be in the form of a pair of counters for each tape being used as an output tape. One counter for each of the tapes contains the ideal distribution for one merge pass, while the second counter contains the total number of actual strings written on each tape. Strings are written onto each tape until the pair of counters for that tape are equal. When all of the counters are equal for one ideal distribution, the ideal distribution counters are updated to the values for an additional merge pass. The flow of the distribution process and the use of the counters is shown for the pre-sort for a four-tape Cascade sort in Figure I.

The second segment of the Cascade sort is a merge sort, during which each pass over the file begins with an N-1 way merge (where N is the number of tapes available to the sort) that continues until the work tape with the least number

of strings is depleted. As each tape is depleted, the way-merge is decreased by one until the pass is concluded by a one-way merge (copying), depleting what was the longest work tape.

Many methods for demonstrating the flow of information through a merge sort have been developed and used, none of them to the satisfaction of this writer. A picture does, however, replace a thousand words, so a Cascade sort is pictured in Figure II using numbers representing the number of strings on each tape at each step of the sort.

The power of this sort technique is derived from the fact that a larger percentage of the file is merged during the most powerful way-merge than is merged during the later phases of the pass; more specifically, for a four-tape sort, 52% of the file is merged during the three-way merge, 36% is merged during the two-way merge and only 12% of the file is involved in the copy portion of the pass. A normal four-tape merge sort is continually performing a two-way merge, giving it a merging power of 2. The Cascade sort for a like number of tape drives has a merging power of 2.3. That is, it reduces the number of total strings by a factor of 2.3 each pass.

A further advantage of the Cascade sort, which is implied above, is that an odd as well as an even number of tape drives, and as few as three, can be used to full advantage.

The sort routines described above assume a read-backward merge sort, which eliminates the need for rewinding during the sort, but which requires the copy portion of each pass in order to reverse the order of the remaining strings on the long work tape. (A read-backward sort must switch the strings from ascending order to descending order on successive passes.) The same Cascade method is available for a read-forward merge, which must, of course, rewind certain tapes during and between passes. This allows the elimination of the copy portion of each pass, as all of the strings are always in ascending order. In order for a read-forward Cascade sort to retain a time advantage over normal read-forward merging, tape rewinding must be a faster process than the tape reading process. All of the comparisons between normal merging and Cascade sorting involving rewinding are based on the Honeywell 400, which has a 3 to 1 ratio of rewind speed over reading speed.

Polyphase Sorting

Further advancement in the efficient use of the tape drives used by a sort has developed from the Cascade sort. This new sorting technique is called the polyphase sorting.

The polyphase sort, like most merge sorting methods, is a two-segment program, a pre-sort segment and a merge segment. Again the pre-sort distributes the ordered strings onto all but one of the available tapes, based upon one of several sequences. The sequences for the various tape drive configurations are as follows:

3-tape 1,1,2,3,5,8,13,21.....(It is noted that this is the same sequence as for a 3-tape Cascade sort. Indeed, a 3-tape read-forward Cascade sort is the same as a 3-tape polyphase sort.)

4-tape 1,1,1,2,2,3,4,6,7,11,13,20,24.....

5-tape 1,1,1,1,2,2,2,3,4,4,6,7,8,12,14,15.....

6-tape 1,1,1,1,1,2,2,2,2,3,4,4,4,6,7,8,8,12,14, 15,16.....

During the merge segment of a polyphase sort, a continuous N-1 way merge is performed. At the beginning of the polyphase merge segment, the N-1 way merge is performed until the tape with the least number of strings is depleted. At this point, instead of switching to an N-2 way merge as in the Cascade sort, an N-1 way merge is continued, merging additional strings from the tapes not yet depleted, with strings from the tape just created. Because this process is continued throughout the merge, there is no point that can be called a complete pass over the file. Instead, there are a series of phases, wherein some strings from a number of previous phases are merged together.

The four-tape polyphase sort example in Figure III shows the effect of this technique through an entire sort. The numbers given in the example represent the number of strings at each of the various phases of the sort.

Comparing the Merges

The power of a polyphase sort is not easily discernible or readily comparable to other sorting techniques, due to the fact that the phases described above cannot be compared directly with the passes of the other techniques. A normal two-way merge sort, for example, processes the entire file being sorted during each merge pass and in so doing, reduces the number of strings by one-half. Another way of stating this is that during each merge pass the length of each string is doubled. This is abbreviated by stating that a two-way merge pass has a power of two. Each step shown in the polyphase sort example processes only a portion of the file. Therefore, while it can be determined from the tables in Figure IV that a four-tape polyphase sort has a power of 1.88 per step as compared with a power of 2.0 for a normal sort using four tape drives, the polyphase is significantly faster because it processes only 62% of the file during each step as compared to the 100% processed during each step of a normal sort.

Figure I contains two tables which show the total number of strings that are merged together for the given number of steps of the merge sort

for four- and six-tape normal, Cascade, and polyphase sorting. The tables also give the percentage of file processed for each step (phase). This percentage is the average for a number of phases in the case of polyphase sorting, the specific values varying slightly from phase to phase.

Further complications in the comparison of the power of the several sort techniques include the internal machine speeds, the amount of simultaneous operation, whether the sort must include tape rewinding, and if so, the relative rewinding speed of the tape mechanism. Figure V relates normal, Cascade, and polyphase sorting as performed on a computer capable of reading backwards and performing all processing at full tape speed. The figures for the polyphase sort have been equivalenced to the same percentage of file processed as for the normal and Cascade sorts.

Figure VI describes graphically the relationship of the power of the three sorting techniques. The graph shows the number of passes over the file required to merge a given number of strings together into one string with four tape drives.

No attempt has been made to date to implement a read-backward polyphase sort. The delay in doing so is caused by the necessity to alternate ascending and descending strings on each of the work tapes during the pre-sort. This alternation of strings destroys the advantage of a variable length string pre-sort whenever some degree of pre-ordering exists in the input to the sort routine. Pure random input data would be sorted faster with a polyphase sort, but experience indicates pre-ordering to some extent exists on the majority of files to be sorted.

Read-Forward Merging

Merge sort routines for computers that allow only read-forward tape operations must rewind a certain percentage of the file between succeeding steps of the merge. A normal two-way merge, for example, rewinds the entire file after each pass of the merge, but, because the file is distributed evenly on two tapes that can be rewound simultaneously, the rewind time for each pass is one-half of the time to rewind the entire file.

A read-forward Cascade sort rewinds a larger percentage of the file each pass than does a normal sort, but retains a time advantage because it does not process the entire file each pass. A four-tape, read-backwards Cascade sort performs a three-way merge over 56% of the file, then must rewind the newly created output tape and the input tape that was depleted. The rewind of the depleted input tape, which delays the operation, is over 19% of the file during the first pass. The Cascade sort then performs a two-way merge of 34% of the file and rewinds the same percentage of the file. The remaining 10% of the file on the third tape becomes input to the next pass and is not processed during the current pass. After the three-way merge on the second and all succeeding passes, the input tape to be rewound includes information used during the three-way merge and two-way merge of the previous pass as well as during the three-way merge of the current pass. This amounts to 56% of the file. Normally, therefore, the Cascade sort processes 90% of the file and rewinds 90% of the file during each merge pass.

A read-forward polyphase sort rewinds the same percentage of the file that it processes each phase. A four-tape read-forward polyphase sort processes and rewinds 62% of the file during each phase. As did the Cascade sort, the polyphase sort depends upon a faster rewinding process than merging process to maintain its full advantage over normal sorting.

Conclusion

The two new merge sorting techniques presented here are now working programs, proven to be the tools for a more efficient computer operation. Cascade sorting provides the efficiency for read-backward operations. Polyphase sorting provides the efficiency for read-forward operations and in the future can provide the efficiency for read-backward operations in cases where the file to be sorted has been determined to be in random order.

There is one hardware prerequisite which should be mentioned, which is necessary to obtain the full advantage of Cascade and polyphase sorting. If reading and writing are to be simultaneous, the sort must be able to read from one tape and write on another in any combination involving the tapes used by the sort.

It is worth noting that improved approaches to the common, everyday computer problems are still being found in an era where the emphasis is on new uses for computers and new designs for computer hardware.

	Ideal Distribution Counters (Total Number of Strings)			Distribution of Strings (Successive String Numbers)		
	Tape A	Tape B	Tape C	Tape A	Tape B	Tape C
One merge pass	1	1	1	1	2	3
Two merge passes	3	2	1	4,5	6	
Three merge passes	6	5	3	7,8,9	10,11,12	13,14
Four merge passes	14	11	6	15-22	23-28	29-31

Fig. 1. String Distribution for a Cascade Sort

Tape A	Tape B	Tape C	Tape D	
14	11	6	0	Output from pre-sort
8	5	0	6	After three-way merge of 6 strings
3	0	5	(6)	After two-way merge of 5 strings
0	3	(5)	(6)	After copy of 3 strings
				End of first pass
3	0	2	3	After three-way merge of 3 strings
(3)	2	0	1	After two-way merge of 2 strings
(3)	(2)	1	0	After copy of 1 string
				End of second pass
2	1	0	1	After three-way merge of 1 string
1	0	1	(1)	After two-way merge of 1 string
0	1	(1)	(1)	After copy of 1 string
				End of third pass
1	0	0	0	After three-way merge of 1 string
				End of sort

Note: All numbers represent the number of strings on each tape at each step. The underlined numbers are the output at each step.

Fig. 2. Cascade Sorting.

Tape A	Tape B	Tape C	Tape D	
13	20	24	-	Output of pre-sort
-	7	11	13	After three-way merge of 13 strings
7	-	4	6	After three-way merge of 7 strings
3	4	-	2	After three-way merge of 4 strings
1	2	2	-	After three-way merge of 2 strings
-	1	1	1	After three-way merge of 1 string
1	-	-	-	After three-way merge of 1 string

Fig. 3. Polyphase Sorting.

Table A -- Four-tape Sorts

Steps	Normal	Cascade	Polyphase
1	2	3	3
2	4	6	5
3	8	14	9
4	16	31	17
5	32	70	31
6	64	157	57
7	128	353	105
8	256	793	193
9	512	1782	355
10	1024	4004	653
11	2048	8997	1201
12	4096	20,216	2209
% of file processed per step	100%	100%	62%

Table B -- Six-tape Sorts

Steps	Normal	Cascade	Polyphase
1	3	5	5
2	9	15	9
3	27	55	17
4	81	190	33
5	243	671	65
6	729	2353	129
7	2187	8272	253
8	6561	29,056	497
% of file processed per step	100%	100%	55%

Fig. 4. Number of Strings Merged.

Table of power of read-backward sorts with a tape limited operation.

Number of Tapes Used by Merge	Power of Normal Sorting	Power of Cascade Sorting	Power of Polyphase Sorting
3 tapes	1.5	1.61	1.80
4 tapes	2	2.30	2.79
5 tapes	2.5	2.94	3.44
6 tapes	3	3.62	3.86

Note: The power of a sort, as used here, is the factor by which the number of strings decreases for each full read time of the file being sorted.

Fig. 5. Power of Three Sorting Techniques.

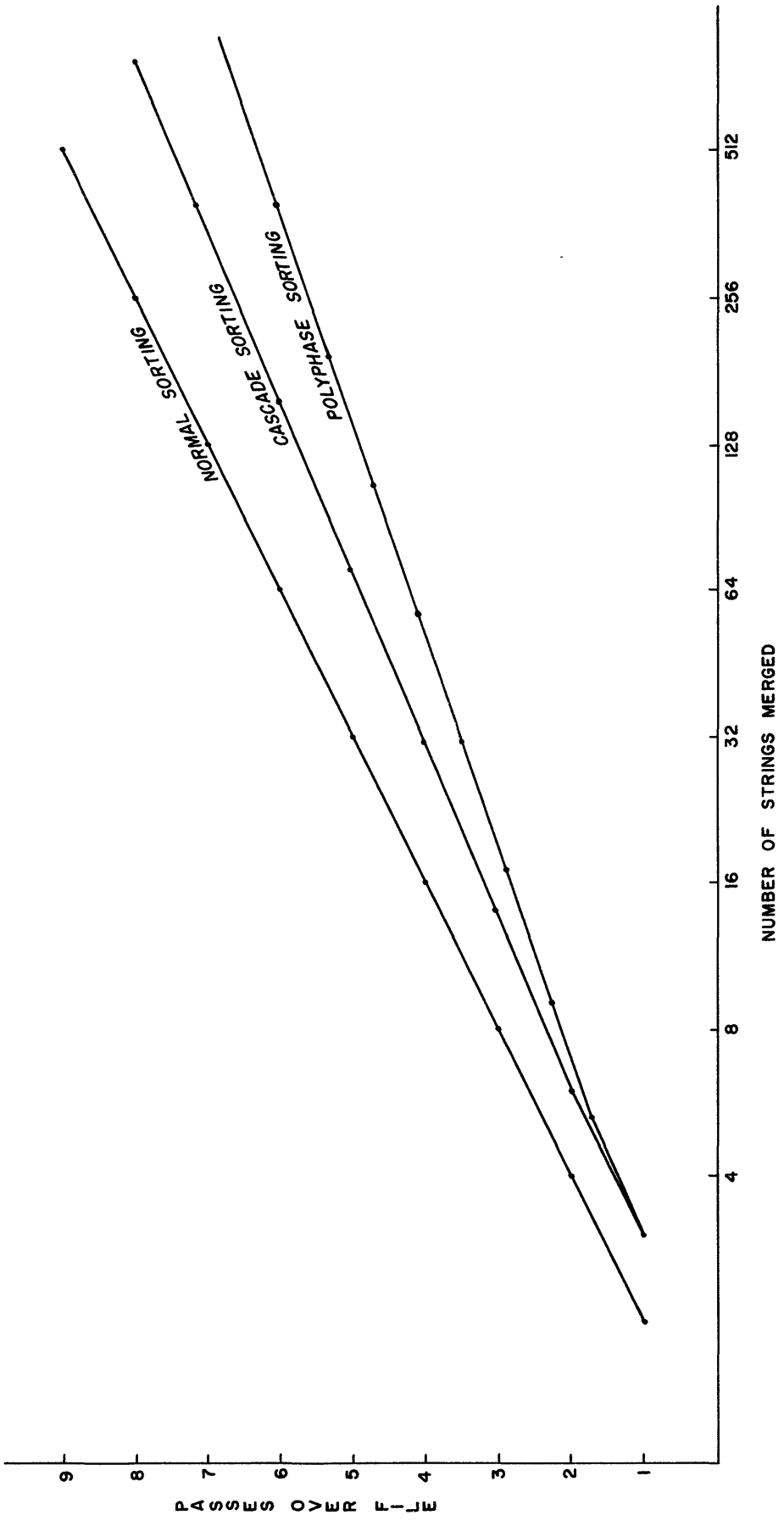


Fig. 6. Comparison of Three Sorting Techniques.

THE USE OF A BINARY COMPUTER FOR DATA PROCESSING

By: Gomer H. Redmond, Manager, Corporate Systems and
Procedures Department, Chrysler Corporation
Dennis E. Mulvihill, PH.D, Senior Consultant,
Touche, Ross, Bailey & Smart

Summary

Considerable discussion has been generated concerning the use of binary computers for purely data processing functions rather than decimally oriented machines. The purpose of this paper is to present a case for the use of binary machines for data processing based on our experience at Chrysler.

Based on experience gained by the Chrysler Corporation, the paper discusses the need for the establishment of a consistency of concept for all phases of problem organization and solution.

Specific advantages inherent in binary machines are pointed out, along with some of the pitfalls which would result if the consistency of concept is not maintained.

In their treatment of this subject, the authors also sound a warning to those concerned with the development and use of generalized business oriented languages that certain abilities of binary machines have not been exploited in these programs.

In their conclusion, the authors state that the abilities of binary-type machines will become more indispensable as management techniques, extant today, become more sophisticated and acceptable.

* * *

There is an unresolved controversy as aired in past issues of the A.C.M. Communications, journals, symposiums, conclaves, and sundry other learned gatherings as to the superiority or inferiority of binary or decimal mode computers in a data processing situation. Both have enjoyed sufficient success in the field to warrant a further look at the controversy.

Let us examine some of the arguments offered for each type of computer. The binary mode machine is said to have the following attributes:

1. Scaling ability allowing information representation to be in any format;

2. Arithmetic circuitry is more efficient in design and speed;

3. Compactness of data in memory;

4. Tape compression factor.

On the other hand, the decimal mode machine arguments center around the following:

1. No transformation required between internal and external representation;

2. Machine language is readily understood by people;

3. Variable length fields are more advantageous to business-type problems.

The above points, from both sides, can be well taken; however, the arguments have been abstracted from the environment in which business data processing is being used, that is, the productive use of data processing as a means to efficient business management. This is the arena where efficiencies and deficiencies of data processing machines can be appraised.

The virtues (and vices) of decimal and binary mode machines have been discussed since man first implemented computer hardware, and the argument will continue until the perfect machine, whatever that might be, is developed and marketed.

I do not intend to discuss the theoretical virtues of either type of computer. I will discuss the controversy in terms of Chrysler's experiences with both types. In the automobile industry, Chrysler has pioneered in the use of EDP. One of the first IBM 702's was installed in our Service Parts Center in 1954. Since then, we have added seven IBM 650's, one UNIVAC I, one UNIVAC File I, and two IBM 709's, one of which has recently been replaced by an IBM 7090. In addition, an H-800 will be installed next spring and four IBM 1401's will be installed in the next few months.

At Chrysler, data processing equipment is selected on the basis of providing the best equipment for the systems in which it will perform. And so it is with all practitioners in the field of data processing. At some point in the game, users must choose between the two modes, binary or decimal. The successful outcome of their venture will depend on their effective use of the equipment regardless of the mode inherent in the selected hardware. It is to this effective use that this paper is aimed, and, by our experience at Chrysler, to prove that more effective use can be made of a binary-type computer for data processing applications of the magnitude of those on the computer in Chrysler's Corporate Information Processing Center (C.I.P.C.).

Chrysler's 7090 installation, C.I.P.C., is used primarily for business data processing, as was its predecessor, the 709. Included in the list of applications are the following major jobs:

Vendor Releasing
17,000 Item, Bi-Monthly;

Hourly Payroll
65,000 Employees, Weekly;

Production Reporting
20,000 Orders Daily;

Warranty Claims & Payments
100,000 Monthly.

We have operated C.I.P.C. for the past year and one-half with the binary mode computer and will continue to rely on this type of machine indefinitely.

Since our frame of reference is the 709/90, our specific examples are in terms of these two computers, but I believe much of what will be said will apply to any binary-type machine of this magnitude commercially available now or in the future.

The earlier binary-type machines, of the 701/704 vintage, could not be used efficiently as data processors. There was a problem in the translation of decimal data to binary for internal processing and vice versa for output operations. Coupled with this, and really quite significant, is the fact that serially-organized machines could not perform data transfer operations without interlocking the central processing units. The desire for business-oriented systems to share computers with their Engineering counterparts, along with other pressures, finally led the various equipment manufacturers to equip their newer binary machines with the ability to perform I/O functions in parallel with internal processing. It is, therefore, the binary-type machine with the ability to perform I/O in parallel with internal processing that will be considered here.

In planning to use any large-scale computer for data processing, it is important that sufficient study be performed in all phases of the system design, programming and installation. In planning the use of a large-scale binary machine for data processing, it is very important, let me say it again: very important, that a consistency of concept be formed early in the system-design phase and maintained throughout all phases to problem solution.

This consistency of concept, if you will, is to think in binary for a binary-mode computer. To exploit the logical power of the binary-type machines, binary techniques must be used. There is a great danger in utilizing a binary-mode computer by applying data processing problems conceived in the decimal mode. Although with their flexibility, binary machines can be used in this manner, it is inconsistent with the organization and logic inherent in the computer and carries a penalty of more involved programming and, in many cases, longer running times.

There is a natural tendency to use decimal representation in record formats; if a binary-type computer is to be used the consistency of concept should dictate that "abbreviated binary" representation be established wherever possible. For straight numeric representation this should afford a thirty per cent improvement over decimal representation.

Although the 709 and 7090 can perform data conversion much more efficiently than earlier binary machines, the time required to convert from decimal to binary and back to decimal can be significant in a large data processing application. Our payroll, in the next few months will increase to over 100,000. An evaluation of the desirable mode of representation for our payroll master file indicated that there was a saving of 34 minutes on the 709 in each payroll run, if the master file were binary rather than decimal.

The representation of information in binary not only applies to magnetic tape and internal operations but to other I/O media. The use of binary-type data, external to the computer, can be quite significant, where portions or all of a standard tab card can be utilized in "Chinese binary" type codings expanding the information content of the card upwards to 12 times its normal alphameric capacity. At Chrysler, this characteristic of the binary mode machine has been most significant in our production scheduling and reporting system. As you may realize, there is considerable amount of information that must be coded for each automobile for production reporting. The many accessories and option combinations must be reported in order to accurately record production. It would be a gross understatement to say that this information could not be represented in Hollerith in a single punched card. Using a binary representation we are able to obtain a maximum of 340 option and accessory codes in 34 columns of a single card. The balance of the card in Hollerith gives us a total of 386 segments of information in a single card. This ability of card stretching is not limited to binary machines--but only recently has this ability been adapted to the decimal machines and then at some added cost of hardware and programming effort.

In many, if not all, data processing problems conditional tests must be made prior to processing input or output data. Binary coding techniques allow single bits to operate as flags or indicators, and single words or strings of bits to contain all of the necessary conditional flags to efficiently operate on the problem at hand. In the decimal machine this ability has been restricted to using individual characters which require several bit positions for a single flag. Quite

recently some decimal machines have allowed access to bits, but even now there is considerable processing and programming effort involved. Bit manipulation inherent in binary machines affords the programmer with the ability to perform Boolean algebra techniques which become significant in terms of economical use of memory, program simplification and job running times.

The bit manipulation ability of binary machines permits the programmer and in effect the program, to perform subtle logical tricks of character, instruction and operation modification in a minimum of time. This ability is particularly useful in housekeeping operations, table construction, address modification and decision-making functions. Very significant gains have been obtained in table construction in our Vendor releasing and production reporting systems. The original system for vendor releasing called for a two dimensional master file. One dimension in parts number sequence and the "other" which was part within major assembly sequence. Originally, it was necessary to sort one file to the order of the other and then sort the extended output for summarization and publication of the requirements.

At the present time, the forecast for all assemblies is placed in memory in a variable length table. This variable length table is controlled by a bit coded basic finder table. The advantage in this case is the effective use of memory in the binary mode machine. We know of no decimal oriented memory that could handle this table. The resultant saving on the 709 from this change was a reduction of processing time from 7 hours to 65 minutes.

Related to the comparison of binary and decimal modes of information representation is the variable and fixed word length characteristics. The binary is typically fixed word oriented. The decimal mode computer on the other hand has been both variable and fixed word length. One argument frequently offered for the decimal mode computers has been in terms of those with a variable word length. Yet, the use of the term "variable word length" with most of these machines may be a misnomer. Many so-called variable word length machines, operating character by character, examined closely are nothing more than fixed word-length machines operating with short "six-bit" words. As a result, the argument on this point is reduced to size-of-word not variable-versus-fixed-word length.

The fixed word binary machine at first glance does not have the address flexibility possessed by variable word length machine. But our experience with the 709 and 7090 proves otherwise, there are in the 709/90 word four addressable segments; the decrement, tag, address and prefix which facilitate packing and unpacking with a minimum cost in instruction time. Most other binary mode machines, I believe, have similiar abilities, along with semi-automatic or automatic masking operations and half-word logic.

There is no question that variability is a desirable attribute in a computer, but only in terms of bits not characters. This value of considering information in terms of bits is most significant in dealing with indicative information.

I would agree that for the representation of quantitative and indicative information in most data processing applications 36 bits is too large a word, but as stated above a "six-bit" word is also too large.

Another powerful ability of the binary machine is its ease of accepting varying formats of coded information. Anything which can be represented by bits can be handled by a binary machine. Thus far, this has been limited because the input/output devices attached to computers were oriented toward the decimal type machine. This ability becomes quite significant when communications schemes link remote operating locations with a variety of equipment to centralized data processors. Format restrictions, in the centralized machines, would require that all equipment be compatible as far as information structure. But with the flexibility of bit coding, a binary machine can be programmed to handle all forms of bit coding. In some of the newer super-scale systems this ability has been further developed to allow the hardware to perform operations in the more popular coding formats, including binary, octal, bcd, bch, telegraph 5, 6, 7 and 8 level coding. The next generation machines will partake of the flexibility now existent in present day binary machines.

The measure of effectiveness for any computer is to a large extent dependent upon the programmer, the program, the compiler and operational system.

In the case of programmers, it is frequently alleged that there is an aptitude difference between those operating on binary mode and those on decimal mode machines. This allegation is carried to the point that the type machine rather than the type of problem is the major determinant in the selection of programmers. Programmers, in the Chrysler context, are those individuals responsible for translating business systems, which are frequently lacking exact definition, to computer systems. At Chrysler, the work performed in C.I.P.C. is business data processing, and therefore, most of our programmers are business systems oriented. The presence or absence of mathematical backgrounds in our programming staff offers no correlation to the effectiveness of resulting programs. One conclusion we have drawn is that programmers first-machine-training has a great deal of bearing on their success as programmers of binary-type machines. Programmers originally trained on binary equipment tend to be better able to adapt to any machine, whereas the converse does not necessarily hold.

In the case of programming, much has been

said with regards to the increased effort required for binary-type machines over decimal machines. Granted that instruction repetoires for the former are usually more extensive than those of comparable decimal machines. The restrictiveness of the instruction lists in decimal machines may necessitate more involved programming for large complex business systems. On the other hand, micro-programming ability inherent in binary machines allows a more precise approach to the solution of complex functions. We realize that subtleties in binary techniques are not readily discernible to the neophyte programmer, thus the learning curve may be somewhat longer, but the results are more significant in terms of programming efficiency.

Significant gains have been made in the field of generalized compilers, both in the scientific and commercial areas. We are heartened by the progress that has been made in the compiler area to date, but are not placing our programming effort "on-the-line" with these until the specifications call for, where possible, a binary approach to problem solution. The user must be wary of sacrificing program effectiveness for programming efficiency.

In the specific area of generalized business compilers, we would like to sound a warning to those of you with binary machines who hope to utilize these to great advantage over your present programming methods. It is apparent to us that, to date, little effort has been expended in the generalized compilers to sufficiently utilize binary techniques which will provide efficient programs such as we have conceived and implemented in lower-level coding systems.

Included in Chrysler's programming standards, are general rules which inhibit the use of generalized sub-routines where tailored routines would better serve to improve program efficiency. The use of true macro-generators which produce effective routines is preferred over the library call and copy type sub-routines.

It is apparent to us that the role of scientific computation in operations research projects is dependent upon an inexpensive, well-structured, data collection system, where data collection is not the major objective but a planned by-product of routine, clerical data processing applications.

Many companies have moved head-long into the field of Operations Research only to find that specific information was not available or could only be obtained at great cost. With the Corporate Information Processing Center at Chrysler we have centralized the flow of information on several major systems. Data is now available on the personnel, production, sales and inventory system in the same format and collected automatically from the routine data processing operations now on the computer. In this respect the utilization of these centrally located files for operation research type analysis along with the proven scientific capabilities of a binary type machine puts at Chrysler's disposal the means to a vastly improved business management system.

Specifically, in the analytical area, Chrysler is currently supplying the Industrial Relations Dept. with personnel and payroll data on the characteristics of the total hourly-rated work force. This information was formerly collected on the basis of a 5% sample. Even with 100% of the data, with less time and effort necessary to prepare and compile the needed data for our negotiations it will be more efficient and factual. The ease of retention and accessibility of historical data also makes the application of other OR projects in the personnel area feasible.

In the Sales area, dealer order and retail sales information which serves as input to our Production Reporting and Scheduling application will be edited, and condensed to serve as further input to a sales forecasting system, which is now in the simulation stage.

It is interesting to note that the data is collected as a by-product of the Production Reporting application and the resultant forecasts will be utilized as input to our Production Programming system, tying together two of the major applications now in our centralized operations.

Other OR projects currently under study are in the manufacturing area, concerned with man assignment and assembly line balancing, and in the financial area where Cash budgeting and Finance Company simulations are in the offing. Both of these studies require extensive data which has been extrapolated from current data processing applications.

Although we have all played lip service to the concepts of management by exception techniques, there is every reason to believe that the data processing function will assume greater responsibilities in the area of management decision and the requirement for large volumes of data output will decline. Furthermore, as data processing applications move from the performance of routine clerical functions to the integration of operations research type solutions of management problems, the computational and logical abilities of binary machines will become more indispensable.

HIGH SPEED PRINTER AND PLOTTER

Frank T. Innes
Briggs Associates, Inc.
Norristown, Pennsylvania

Summary

The Model 1063 High Speed Printer and Plotter is a magnetic tape fed device for high speed plotting up to 6000 points per second in ten simultaneous plots, at the same time it prints annotations for grid lines and draws the grid lines, all with the output paper moving at 10 inches per second.

The machine may also be used simply as a high speed printer. In this mode, it can print 4000 lines per minute with 100 characters per line.

Design Philosophy

General

The general design philosophy for this machine was to produce a highly flexible off-line device which would minimize programming requirements and machine time for the IBM 7090 computer which generally prepares the input tapes. Since it was early recognized that the required generality would require the machine to have quite substantial high speed printing capabilities, it was specified that the machine be capable of operating simply as a printer. The success with which this philosophy has been implemented is indicated by the fact that soon after delivery the customer modified the machine to accept tapes produced directly by his analog-to-digital format conversion equipment without any computer processing; thus it operates valuably as a semi-quick look recorder. Likewise as a printer, it is used on a routine basis to print tapes prepared for conventional IBM printing equipment. The customer is now making minor additions so that the machine will perform binary to octal conversions; in general this will completely eliminate such operations on the computer.

Detailed Design

The detailed design naturally stressed reliability and ease of maintenance using readily available components. Since this machine was built on a limited budget, cost was a very important object also. Commercially-available digital modules are used in the control logic areas. In the other parts of the machine where circuits are used by the hundreds special purpose modules were built. Worst case design with extreme derating of components was employed in combination with rather eclectic logic.

Each circuit area was examined from the point of view of least cost. As a consequence, the machine is a mixture of diode logic, diode-transistor logic, and resistor-transistor logic.

Overall Logic

A very general block diagram of the machine is shown in Figure 1. There, it is apparent that the machine is really two machines, having a common input and output. When printing only is to be accomplished, only the print section is used. When annotated plotting is to be done, the machine starts off as a printer, allowing the title block to be composed at the programmer's discretion. When the first record containing data points occurs, then the machine switches control of printing to the plot section where it remains until the graph in question is complete. With this mode of control, it is a simple matter to coordinate the annotation data with the grid lines. If it is desired to establish a time reference with a grid line, the necessary command is placed on the tape next to the appropriate data point; if it is desired also to annotate this line, then the alphameric data with a print command are placed there also. The machine then operates to route the data in such fashion that the required alphameric data is waiting for the print command when it occurs. Subject to minor programming restrictions, all these operations are done without interfering with the steady flow of data required for a time history plot.

Output Device

The output device for this machine is a Hogan Laboratories Model HPP-110 multiple stylus recorder.

This unit is a facsimile type recorder wherein a mark can be made on moist electrolytically-treated paper by application of an appropriate current. The HPP-110 has 1024 writing styli which press 11-inch wide paper against a reciprocating steel strip with paper moving at five or ten inches/second. In the process of plotting or printing currents are applied through the styli to the bar; the electrolytic process involved removes metal from the bar and leaves it in the paper; the duration and magnitude of this current are such as to produce highly uniform elementary dots approximately one one-hundredth inch in diameter. Wear on the stylus assembly is abrasive only; currently, about 30,000 feet of paper have been processed without degradation of marking capabilities.

Output Circuits

Requirements

The HPP-110 is provided with its own direct-coupled power amplifiers one each for the 1024 styli. It is readily apparent that some means must be supplied to provide signals of the proper magnitude and duration for the power amplifiers. Plotting requirements call for 1000 such circuits; detailed consideration of the logic revealed that separate printing circuits must be supplied also in addition to auxiliary inputs for drawing of grid lines.

The specification of two paper speeds also leads to two signal duration requirements. Since the same elementary dot must be produced at either speed, provision had to be made for inexpensive variable duration control, explicitly for 0.5 milliseconds and for 1.0 milliseconds.

Circuits

The actual output circuit card is shown in Figure 2. The basic element is the NOR flip-flop with a 2-input diode gate on the set input. The upper row of five units provides for five out of the 1000 possible plot points with the diode gate forming the second level of a 1000 point binary decoder.

The lower four units correspond to the first four elements in the seven styli that are used for a column of printing: three corresponding units on the next card provide a total of seven, which gives the space dimension to the 7x11 space time matrix used to form alphameric characters. The diode gates on these flip-flops are used for column selection in conjunction with a column counter.

In the general case, the outputs of two flip-flops are buffered together to drive the power amplifier. Other inputs to the diode buffer provide grid line drawing capabilities under either tape or patchboard control.

Signals to energize particular styli are of nominal ten-microsecond duration, either from the print section character generator or from the plot decoder. Signals to de-energize are applied to the reset inputs. In the case of printing, the column counter selects a particular column whose styli are energized depending upon the state of the character generator. The flip-flops corresponding to this column are then turned off by an unconditional output from the column counter fifty columns later; since the column counter operates at either 50 or 100 kc, the required 0.5 or 1.0 millisecond durations are automatically generated.

A similar operation determines the duration of the point to be plotted; however, this is done under patchboard control since it may be necessary to plot line segments rather than elementary dots, particularly when the plotting rate is low.

Plot Section

Plot data and associated control characters are stored in small coincident current memory buffers from whence information is transferred to the plot section under control of the plotting rate clock. Ten bits out of the twelve possible in two IBM tape frames are used to specify a point to be plotted. The occurrence of a plotting rate clock pulse triggers the readout of data from the memory, a read cycle being required for each tape frame. If the two frames are recognized as a data point, then the ten bits are assembled in the plot register from whence they are decoded and used to set the appropriate output flip-flop. If a control character is recognized, it is decoded and the appropriate action taken; at the same time another pair of memory read cycles is initiated until a data point is found. If more than one plot is programmed, the above process is re-

peated until the preset number of simultaneous data points has been extracted; although "simultaneous" data points are actually plotted at 20 microsecond intervals, the distance between such points on the output graph (0.2 mils) is naturally not distinguishable to the naked eye.

As noted above, provision is made so that various mark lengths can be selected by the operator. Likewise, the operator must specify number of simultaneous plots and plotting rate. Plotting rate capabilities vary from 1000 to about 16 points per second per curve, allowing great latitude in expanding or contracting the time scale with the same input tape and paper speed.

Print Section

A block diagram of the print section is shown in Figure 3: its essential parts are the recirculating buffer, the character generator, the column counter and the trace counter. The heart of this section is the character generator which stores, in wired form, the particular selection of dots in the 7x11 dot matrix corresponding to each of the 56 characters. The character generator is divided into eleven sections corresponding to the eleven rows in the matrix; the rows are selected by the trace counter which advances one step for each scan of the column counter across all 100 columns. The recirculating buffer is synchronized with the column counter to select, on the basis of the IBM BCD codes stored therein, the proper group of dots from the character generator for each character on each trace. Thus, during the first trace, the topmost portion of each character in each column is placed on the paper with the process repeating until the full character is printed.

Character generator outputs drive all columns simultaneously with the column counter being used to determine the proper column. Normally characters are printed in succeeding columns starting with the first, in the order in which they appear on tape. Column counter outputs are, however, also brought out to a patchboard where characters may be rearranged in any desired manner and where provision is also made to repeat any character once. This facility makes it possible to minimize BCD data on tape during plotting, space codes in effect being supplied by the patchboard.

A special feature of the print section lies in the brute force design of the character generator, there being one specific diode corresponding to a given dot in a given character. This approach to the character generator was the cheapest under the circumstances, but in combination with great redundancy in the 7x11 matrix character, it has the merit that the loss of a diode is scarcely noticeable even to the close observer.

Input and Tape Organization

Tape

The tape is organized generally in records, either straight BCD for printing only or combined BCD and Binary for annotated plotting. The IBM 36-bit binary word is divided into three 12-bit groups with 10 each required to specify a data point. Control characters are provided on the tape and indicate how data is to be interpreted and control its routing either to plot or print input buffers. A record may include up to 71 data points and up to 100 BCD characters. Tape operates at 150 inches per second with 200 bits per inch.

Plot Section

Since the tape operates start-stop in order to provide flexibility in the machine as regards paper speed and plotting rate, then two buffers must be supplied in the plot section to guarantee the required steady flow of data; in general, one is being filled while the other is being emptied. Circuitry and logic would allow the processing of 10,000 data points per second; however, tape characteristics and buffer size limit the maximum to about 6000.

Print Section

Two buffers are also provided in the print section in addition to the recirculating buffer discussed above. If the machine were to operate simply as a printer, these buffers would be unnecessary. Operation of the machine as a printer-plotter with generality and ease of programming appears to demand the presence of the two print input buffers: other arrangements were considered and found to place severe limitations either on the program or on the performance of the machine.

Reliability and Performance

Reliability of the machine has been excellent. Precise statistics are available only for a period of 300 hours. During this period, there has been but one component failure, a transistor. When the number of components is considered - 9000 transistors, 12,000 diodes, about 40,000 other components - this is quite a good record.

Likewise, the performance has been excellent; except for the transistor failure noted above, maintenance has been limited to the routine care necessary to the proper performance of the tape transport and occasional adjustment of the recorder mechanism.

Acknowledgments

I wish to acknowledge at this time the contributions of G. Conklin, R. Hench, R. Hibbs and C. Spindler of the General Electric Company. Without their vision and support, this machine would not have existed.

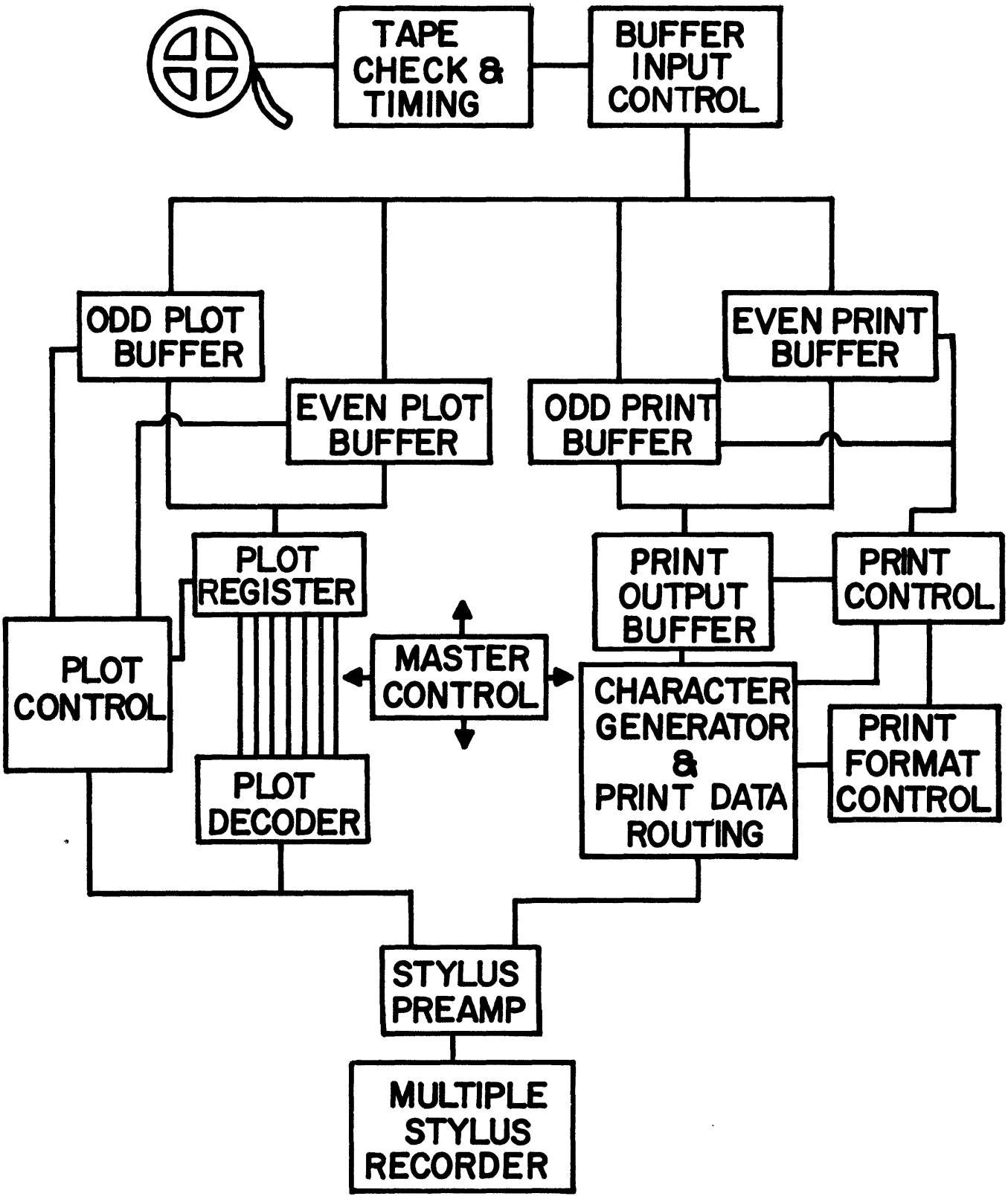


Fig. 1. Block Diagram.

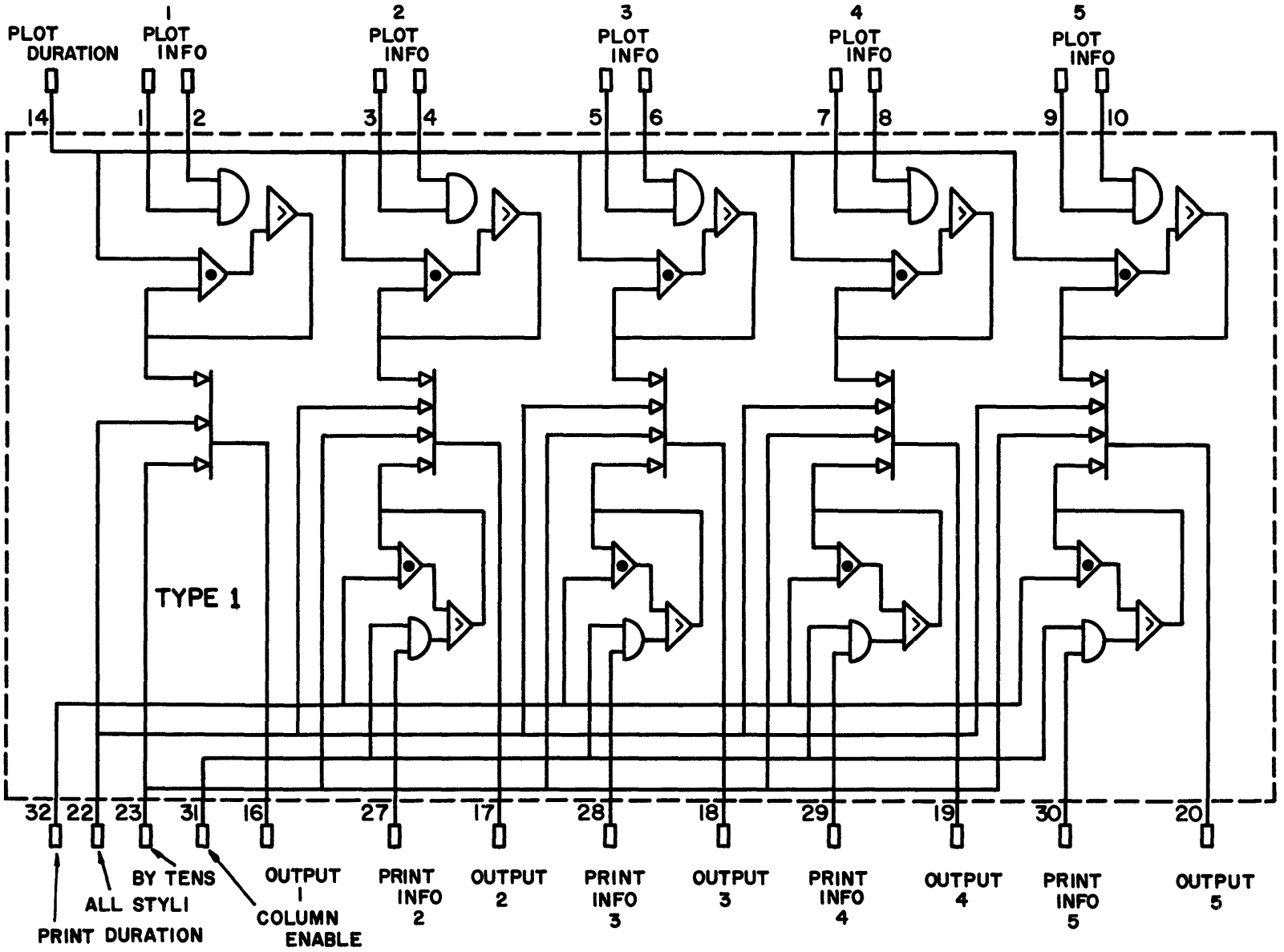


Fig. 2. Output Circuit.

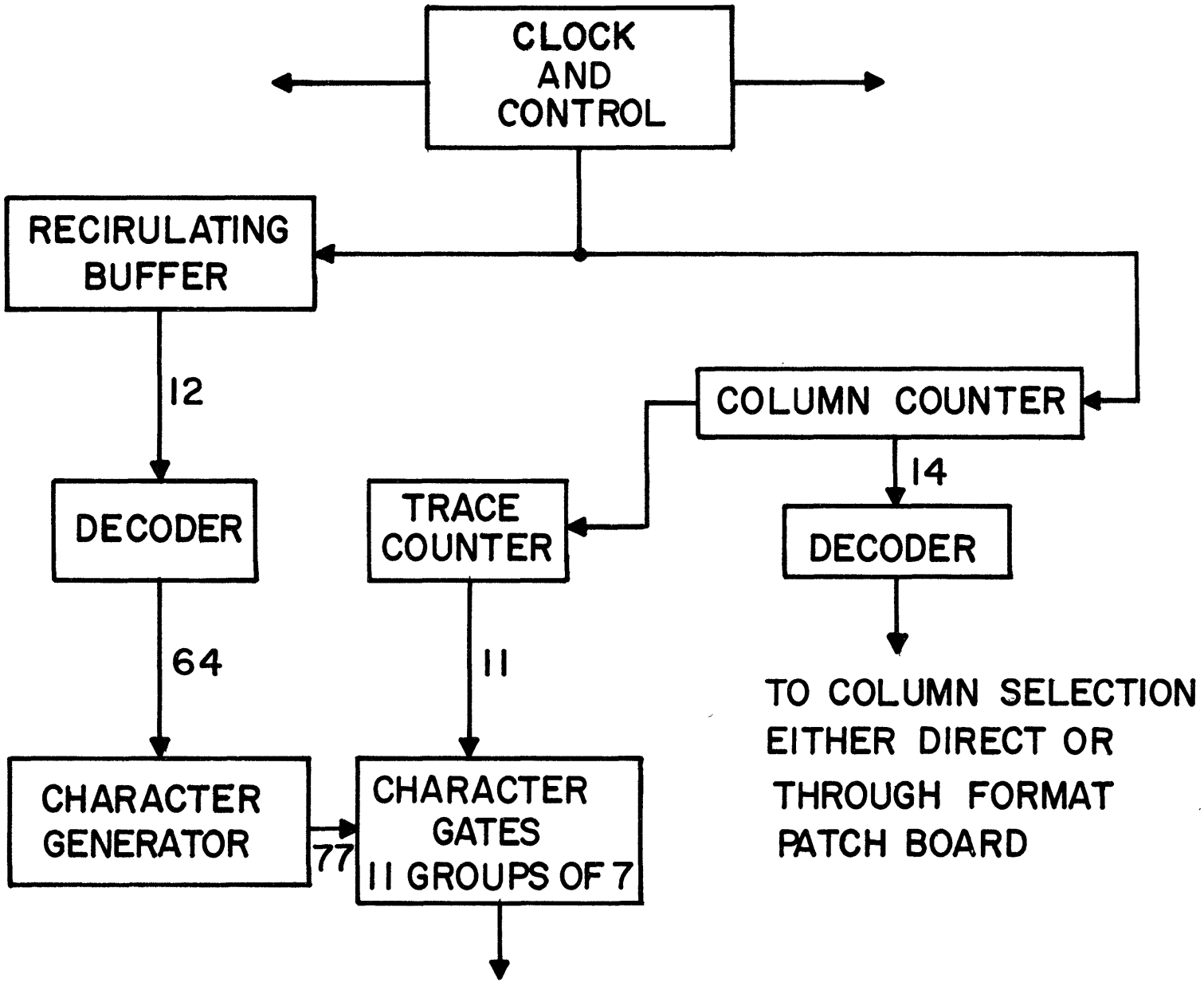


Fig. 3. Print Section.

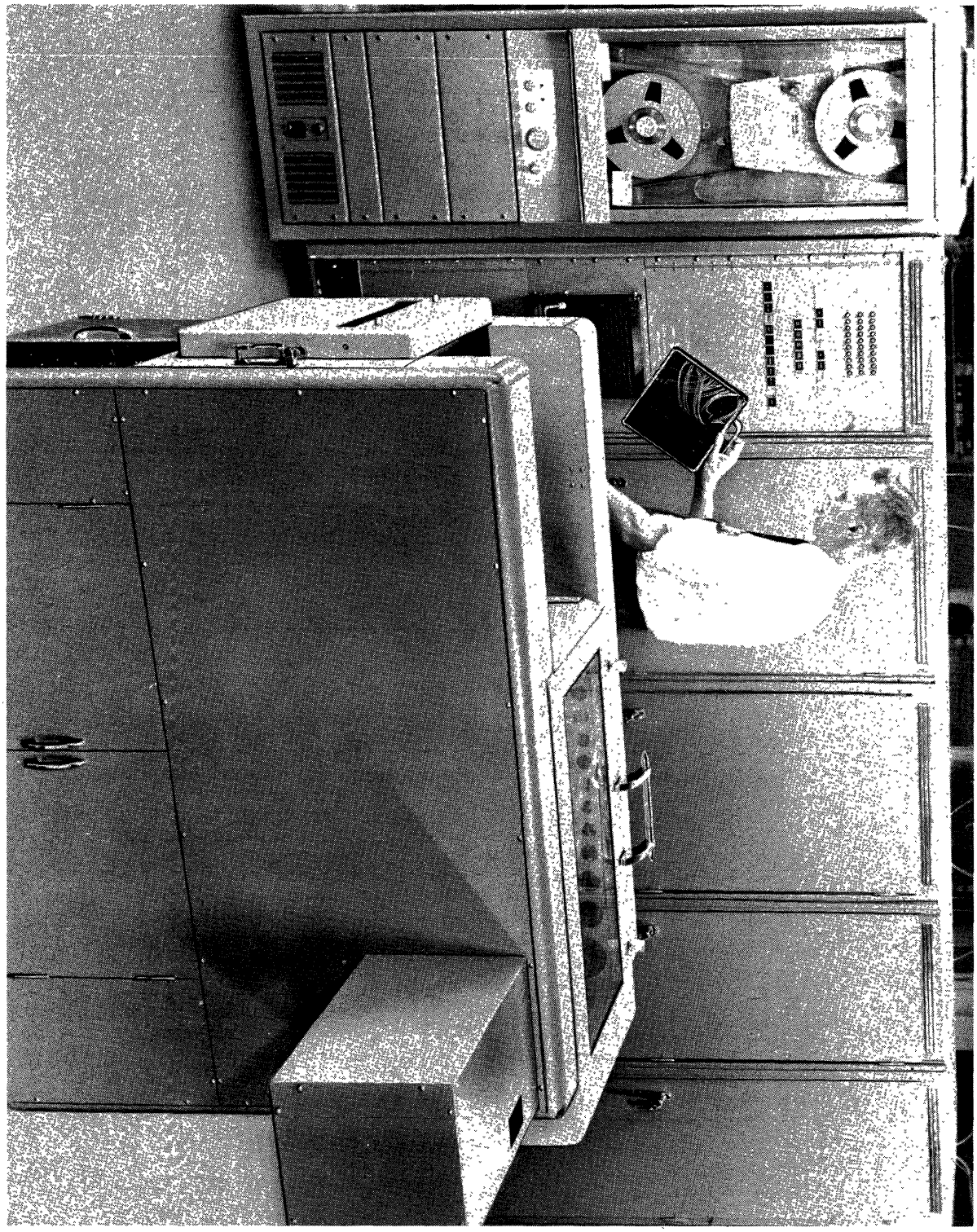


Fig. 4: General View of Printer Plotter.

A DESCRIPTION OF THE IBM 7074 SYSTEM

R. R. Bender, D. T. Doody, P. N. Stoughton
Product Development Laboratory, Data Systems Division
International Business Machines Corporation, Poughkeepsie, New York

Summary

A new data processing system, the IBM 7070[1], was described at the 1958 Eastern Joint Computer Conference. Recent progress has resulted in the creation of an expanded family of 7070 systems, exemplified by the announcement of the IBM 7074 system.

The 7074 represents a dramatic new approach to data processing system growth, and is the second major step in the 7070 data system family. It is not an entirely new system, but rather an improvement within the 7070 framework. It enables a customer whose workload has outgrown his 7070 equipment to upgrade his system over a weekend, thus achieving multiplied performance without reprogramming and without excessive disruption of his operation.

Specifically, the increased performance is achieved by use of:

1. A new, high-performance arithmetic and program unit called the 7104 high-speed processor.
2. Improved-performance storage units, the 7301 models 3 and 4, which operate on a four-μsec cycle instead of the six-μsec cycle used with the basic 7070.

These units are substituted for their counterparts in the 7070 system.

Compared to a two-channel 7070 system using 729 IV tape drives, a 7074 of the same configuration has the following performance characteristics:

Internal performance on commercial work	6 x 7070
Thruput or job performance on commercial work	2 x 7070
Floating point performance on scientific work	10 x 7070

Internal performance is a measure of instruction-execution time. It is measured on the basis of the mix of instructions executed in a group of programs considered typical of commercial applications.

Thruput or job performance considers, in addition to instruction-execution time, the time expended in magnetic-tape input/output operations. A typical mix of commercial jobs--including sorting, merging, high and low activity-file maintenance, and editing--provides the basis for this comparison.

Floating point performance is measured on the basis of a typical group of arithmetic and logical instructions encountered in many scientific problems. It is essentially a measure of internal speed.

Functional Units

Physically, the 7070 family is made up of the following functional units which are packaged in IBM standard modular system (SMS) frames:

	7070	7074
Arithmetic & processing unit	7601 (2 modules)	
High-speed processor		7104
Storage, 6-μsec	7301, 1 & 2	
Storage, 4-μsec		7301, 3 & 4
Core control and power distribution	7602	7602
Basic timing & control	7600	7600
Tape control unit	7604	7604
Tape transports	729 II or IV	729 II or IV

The 7601 arithmetic and processing unit and the 7104 high-speed processor perform arithmetic, logical, and other stored-program operations under the control of a single-address type of instruction. Some other features are: ninety-nine indexing words; variable field length by the use of field definition; automatic block transmission of data within core storage; automatic priority processing; extensive checking; and simultaneous read, write, and compute.

The 7301 storage units are available in models of 5000 or 9990 words. They provide parallel access to ten digits (one computer word) at each storage reference.

The 7604 tape-control units provide for transmission of data between tape storage units and core storage. Two independent data channels can be provided by each 7604 unit.

The 729 tape transports are available in two models. Model II can operate at data rates of 15,000 and 41,000 six-bit characters per second. Model IV can operate at data rates of 22,000 and 62,500 six-bit characters per second. Up to forty tape transports, in any combination of models, are available on the system.

The above listing is by no means exhaustive. Many other devices -- including punched-card devices, printers, and manual inquiry stations -- are also available.

These units are the building blocks of the IBM 7070 family. Proper selection of processor, memory, and tape drives provides the ability to tailor a data processing system to a wider variety of customer requirements (both commercial and scientific) than ever before possible.

System Growth

The IBM 7074 system may be ordered directly from the factory, or it may be "grown" from a 7070 in the customer's office. The necessary changes can be made by a team of field engineers over a weekend.

Referring to Figure 1, the 7104 high-speed processor is substituted for the two 7601 modules which are removed and returned to the factory.

The 7301 storage unit is converted from a six-μsec cycle to a four-μsec cycle by a change to high-speed circuitry.

One slide, containing storage controls, is removed from the 7602 core control unit and returned to the factory. New storage-control circuits are provided in the high-speed processor.

Program Compatibility

The 7104 high-speed processor uses the same instruction set as the 7601, although it processes individual instructions three to twenty times faster.

Some examples are:

	7070	7074
One-digit true add	48 μsec	10 μsec
Ten-digit true add	72 μsec	10 μsec
Multiply (10-digit multiplier)	924 μsec	56 μsec
Conditional branch	36 μsec	6 μsec
Unconditional branch	24 μsec	4 μsec
Floating add	212 μsec	16 μsec
Floating multiply	1019 μsec	60 μsec

Since the instruction formats are identical, programs written for the 7070 may be used on the 7074 without change. Furthermore, they will operate at full efficiency on the 7074.

This compatibility is important for rapid and simple change-over from 7070 to 7074. In addition, all 7074 customers -- newcomers as well as those changing over from the 7070 -- have at their disposal the entire 7070 program library of the GUIDE organization, and IBM applied programs for the 7070.

Processor Organization

The 7104 high-speed processor, like the 7601 arithmetic and processing unit, operates on the basis of a word of ten decimal digits and sign. Coding is 2 of 5, so that a word consists of 53 bits. Sign is plus, minus, or alpha and is represented by three bits in 2 of 3 code. Alphanumeric information is represented by two decimal digits, so that an alphanumeric word contains five characters, while a numeric word contains ten digits.

When written on magnetic tape, an alphanumeric word fills five six-bit characters. Numeric words are written as ten six-bit characters except that up to five high-order zeros are eliminated. This makes for very high tape efficiency.

The 7104 high-speed processor differs from the 7601 arithmetic and processing unit in that the 7601 performs arithmetic operations in a serial-by-decimal-digit manner while the 7104 performs full-word parallel arithmetic.

In the 7601 (see Figure 2) each digit is moved through the adder in one four-μsec cycle and is stored back in the arithmetic register on the following cycle, during which the next digit is moved through the adder. A full ten-digit add requires eleven cycles or 44 μsec for completion (assuming that recomplementing is not required). To this must be added instruction and operand access time as well as indexing time if required. Total time for a ten-digit true add (not indexed) is 72 μsec.

In the 7104 (see Figure 3) the full-word adder cycle requires two μsec for completion. Instruction and operand access time results in a total time of ten μsec for the nonindexed add instruction. This time is valid for any field size up to ten digits. Thus, add speed has been improved from three to seven times, depending upon field size.

Items of interest are:

1. Skew registers which provide the functions of field control and shift.
2. Three accumulators which provide speed in floating point operations.
3. Validity checking on all buses.
4. Complete program compatibility with 7070 (uses 7070 instruction set).

The information bus is one computer-word wide (ten decimal digits and sign = 53 bits). The address bus is four digits (20 bits) wide, and the arithmetic buses are eleven decimal digits wide.

Circuits and Packaging

The high arithmetic speeds are made possible by the use of saturating-drift-transistor NOR circuits. Packaging is accomplished in a new package known as the SMS twin card, which provides over three times the density of logical elements achieved in the 7601 processor of the IBM 7070. This density permits the 7104 to contain in one module all of the logic previously packaged in two and one-half modules.

Figure 4 compares the new SMS twin card with the SMS single card used in the 7601. Up to 44 transistors may be packaged on one twin card as compared to a maximum of eleven on the single card. The use of NOR circuits further increases the logical density in the SMS twin-card system. Vertically mounted components of the twin cards provide more efficient cooling. The component tips are welded to the bronze support clips at the upper end, and are soldered to the printed wiring of the card at the lower end. The bronze support clips contribute to cooling by providing a heat-sink effect; these clips also provide an additional dimension of modularity for automated production and for field repair of cards. Support-clip sections can be stocked and card repairs made in the field by replacement of clips, thus contributing to more economical maintenance.

Figure 5 shows additional details of the SMS twin card.

Cards are mounted in an IBM standard modular system (SMS) frame, which contains two slides, each composed of two pages. Each page contains four chassis, each of which in turn can contain 100 SMS twin cards.

Figure 6 shows an SMS sliding-gate module, covered. One such module contains the 7104 high-speed processing unit, and measures 29 1/2 in. wide by 56 in. deep by 69 in. tall.

The sliding gates pull out toward the front as shown in Figure 7.

Each gate opens into two pages in which are mounted the SMS single or double cards. The pages or gates are accessible for service from both sides. Covers over the cards contain the flow of cooling air.

Figure 8 depicts the organization of a page or gate of the module. The four chassis, each of which can contain 100 SMS double or 200 SMS single cards and a number of edge connectors, are shown from the rear or panel-wiring side.

It is the SMS system which makes possible the modular growth from the 7070 to the 7074. Replacement of one or more of these frames with functionally similar units of higher performance is possible without re-engineering every unit of the system.

References

1. For a description of the IBM 7070, see "The IBM 7070 Data Processing System" presented by Robert W. Avery, Stephen H. Blackford, and James A. McDonnell at the Eastern Joint Computer Conference, December 1958.

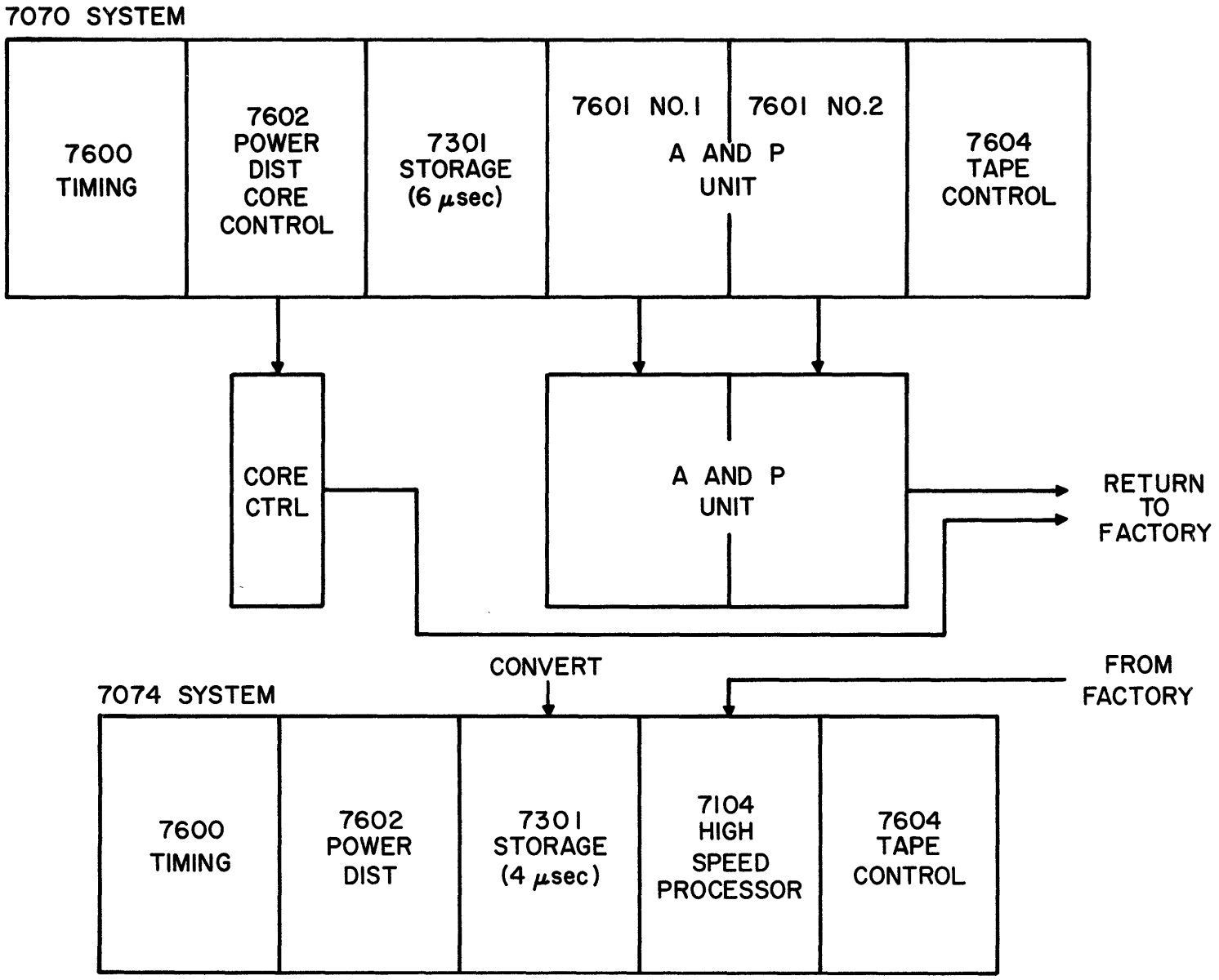


Fig. 1. System growth.

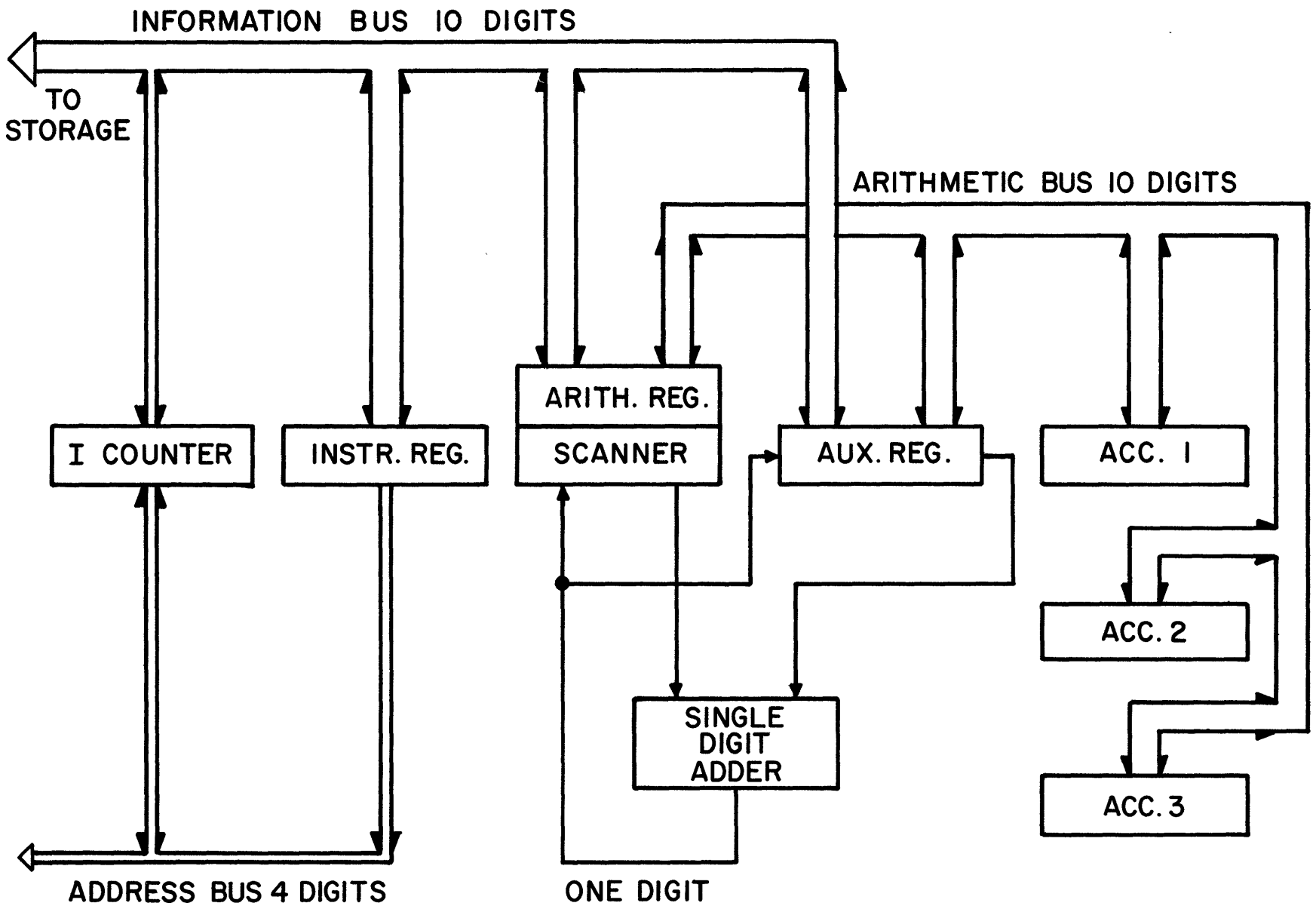


Fig. 2. 7070 information flow.

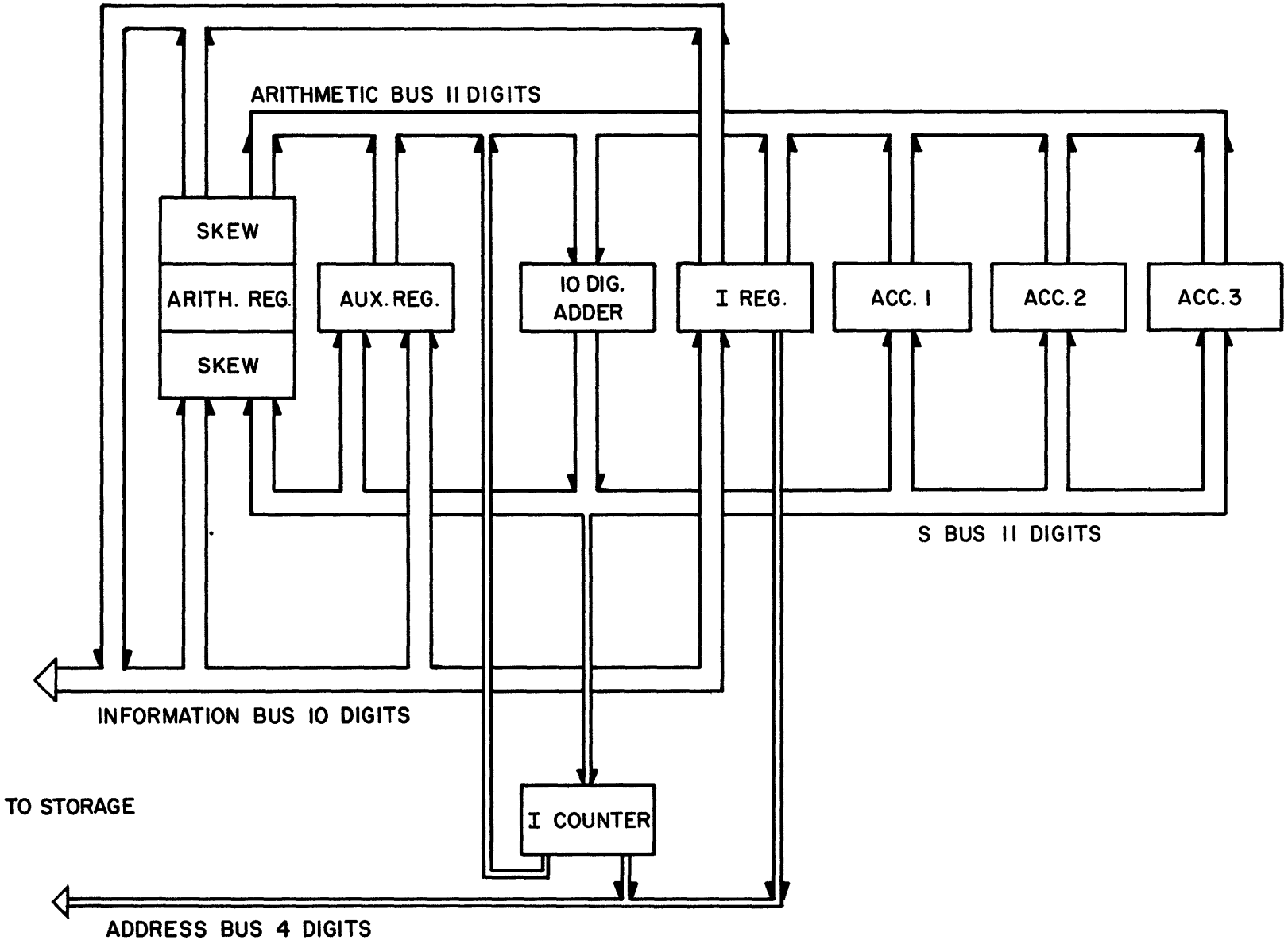


Fig. 3. 7074 information flow.

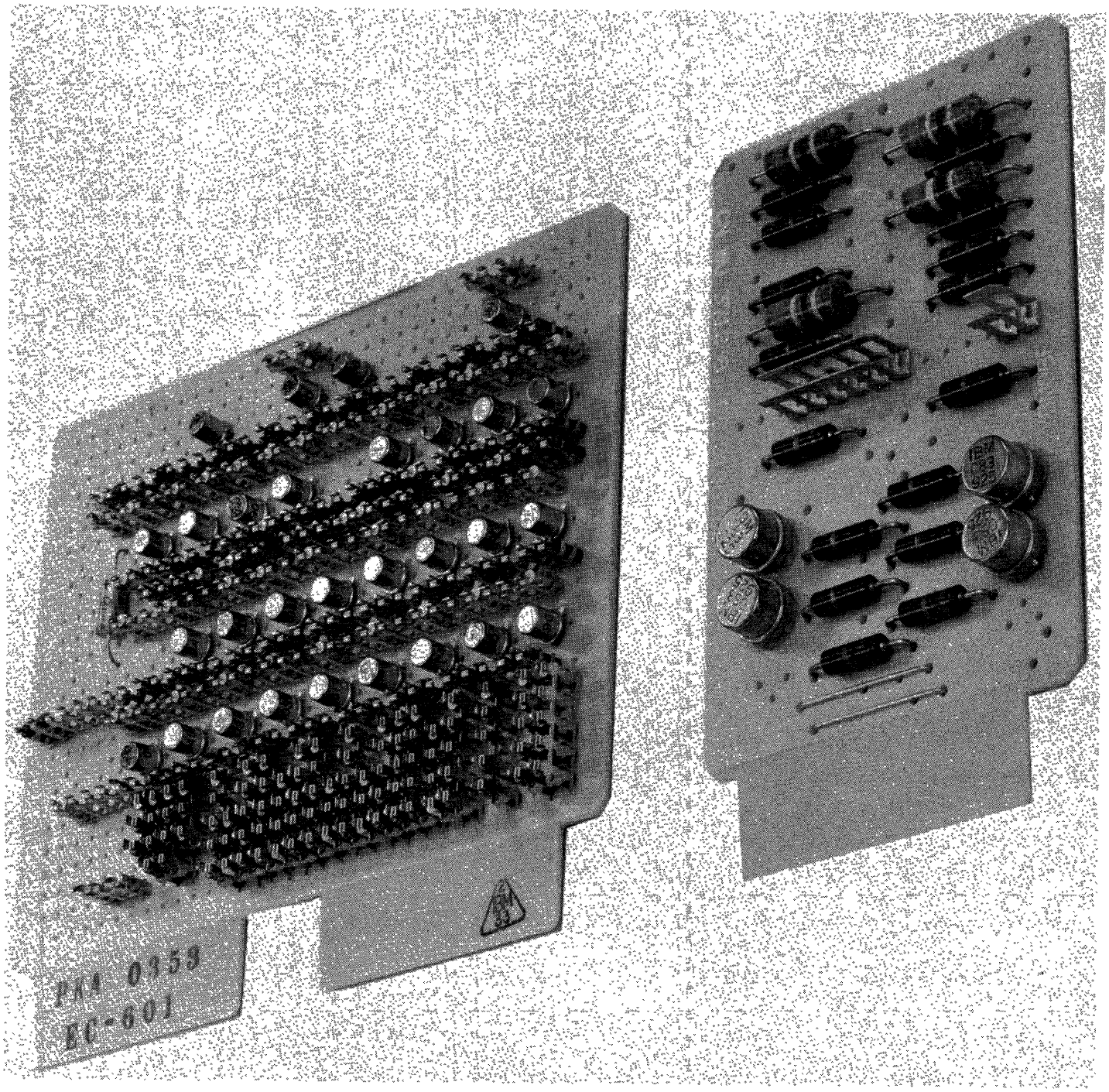


Fig. 4. SMS twin card and SMS single card.

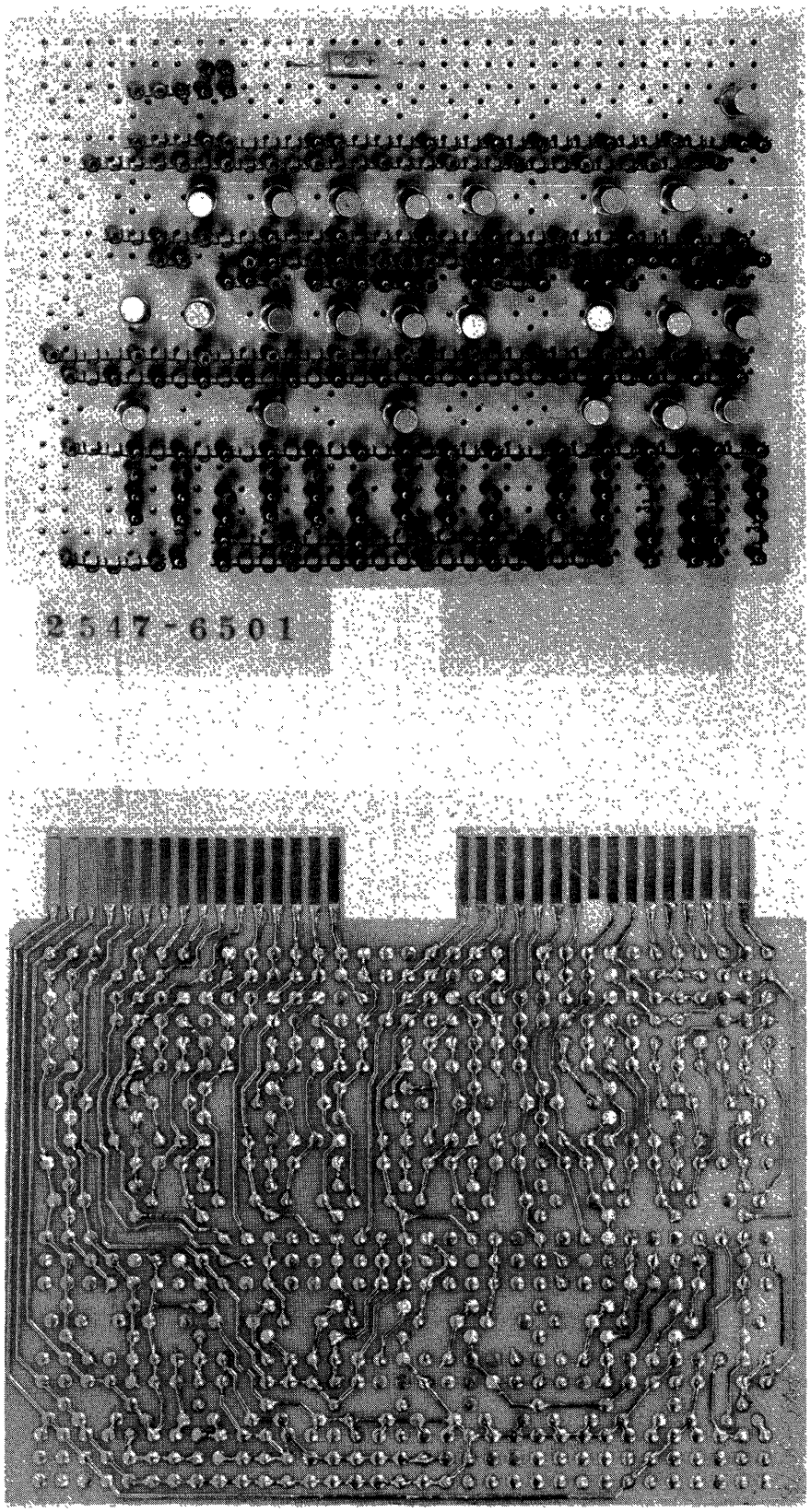


Fig. 5. SMS twin card detail.

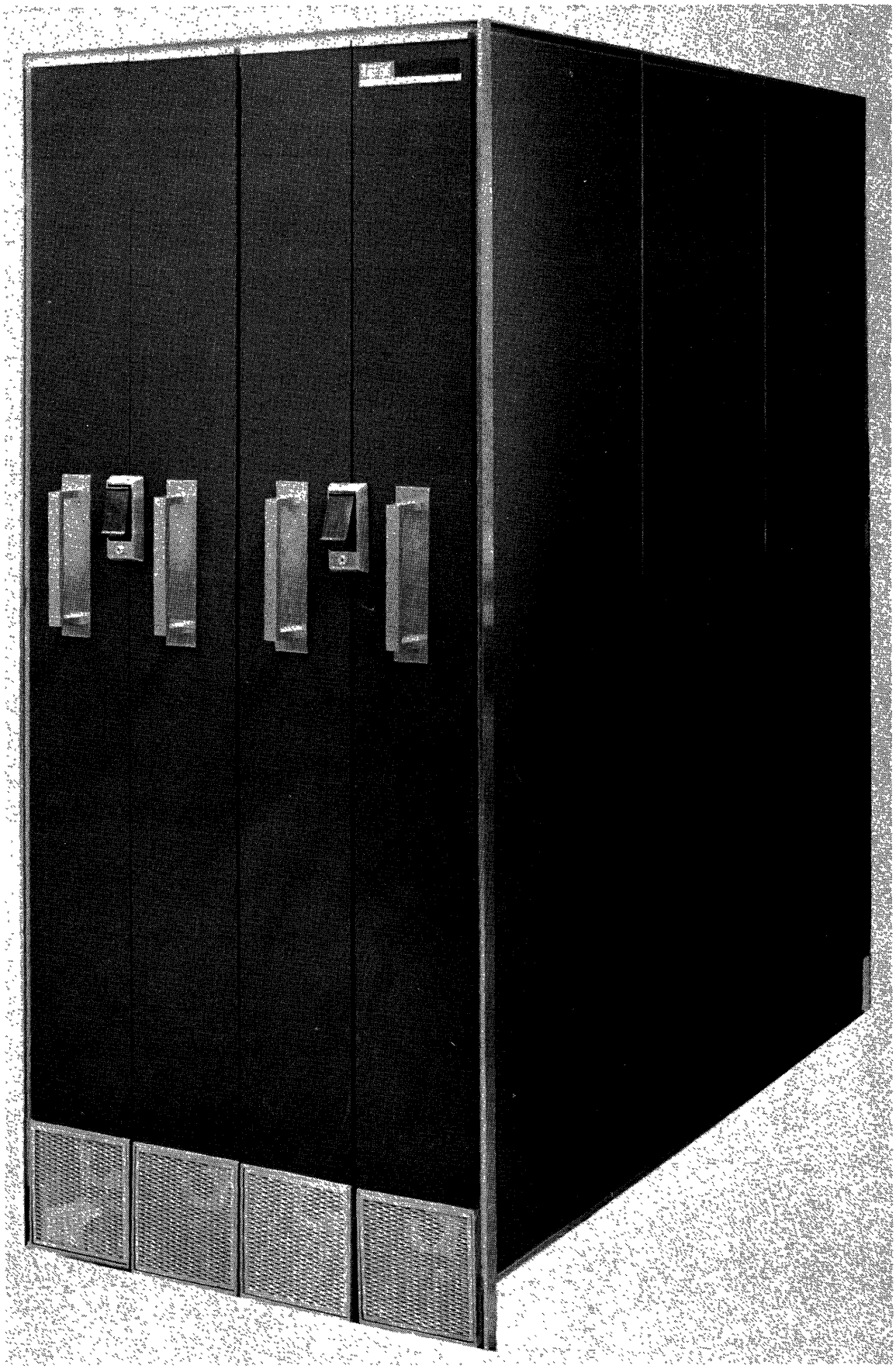


Fig. 6. SMS functional module.

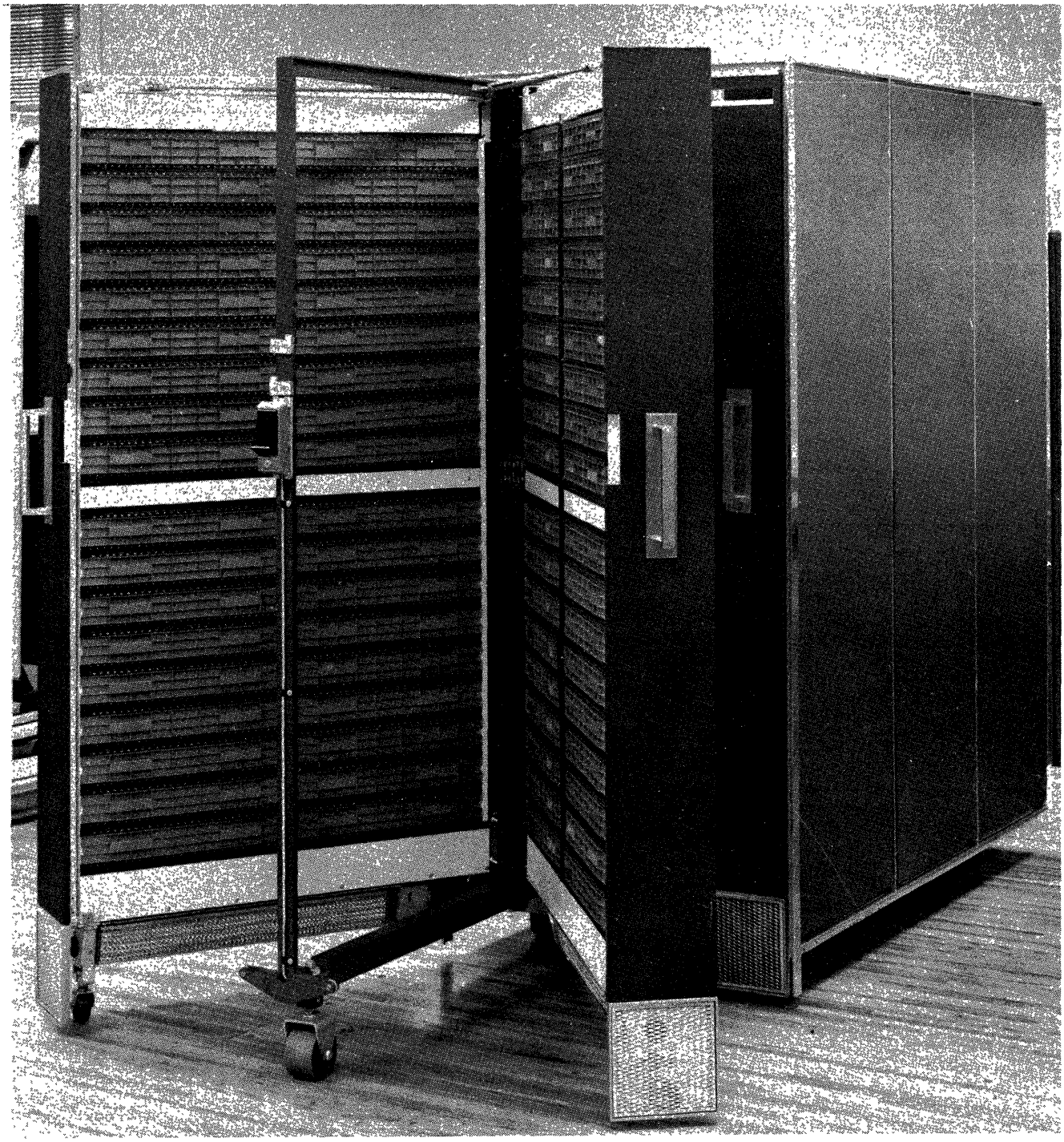


Fig. 7. SMS module with slide out.

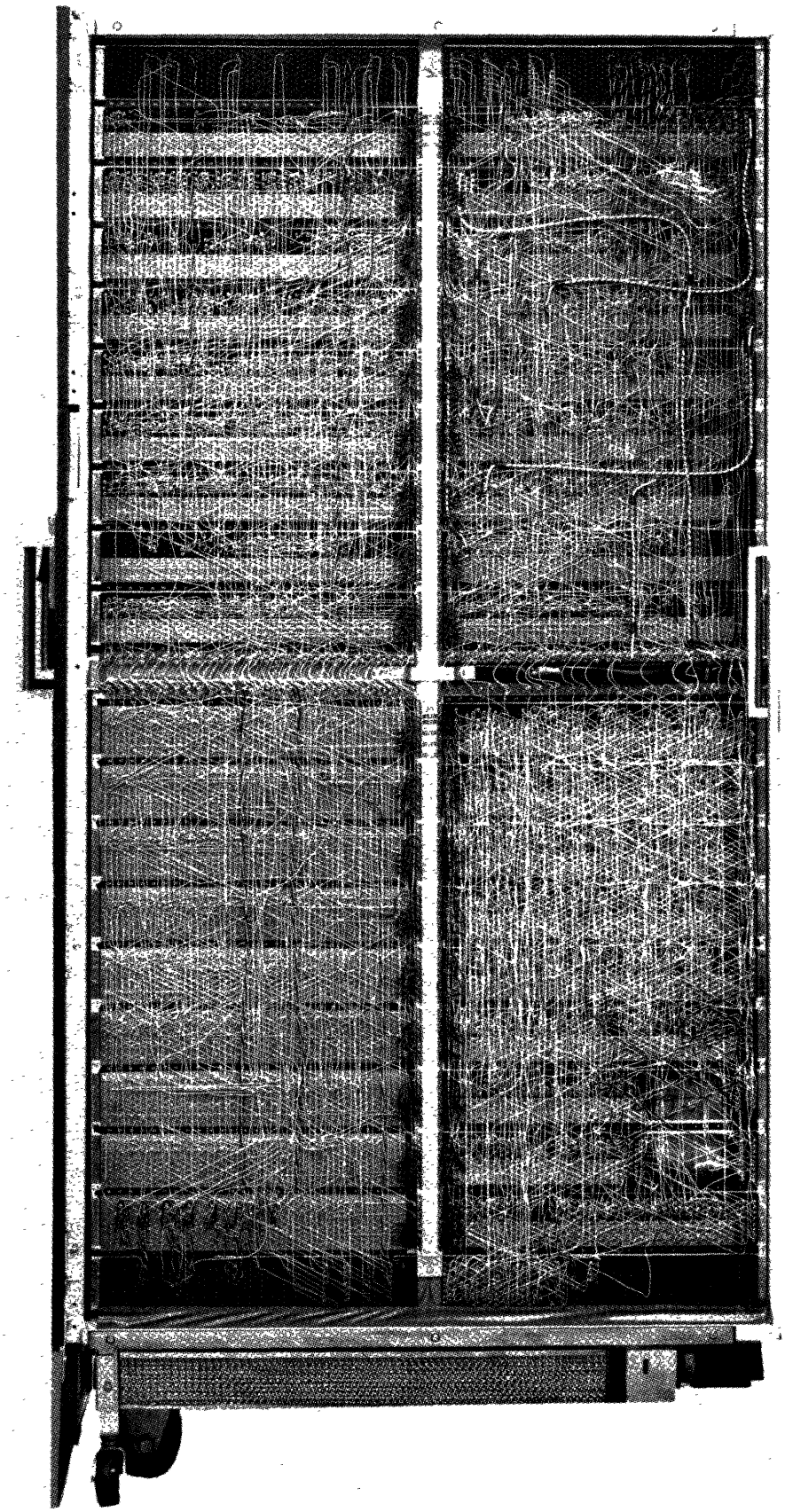


Fig. 8. SMS module page – rear view.

THE RCA 601 SYSTEM DESIGN

A. T. Ling and K. Kozarsky

Electronic Data Processing Division
Radio Corporation of America
Camden, New Jersey

One of the differences found among computer systems is whether emphasis is on word orientation or on character orientation. Pertinent design considerations not only include memory depth, but also capability for handling symbol-controlled operations and relative speeds of word and character operations. Emphasis placed on either word or character orientation tends to dictate a principal area of effective application. An important objective of the RCA 601 System is to perform efficiently over a very wide application base and economically combine the speed of parallel word processing with the logical flexibility of variable character operations.

The RCA 601 System is a generic name which, properly speaking, refers to a class of systems. This is due to a very generalized design approach which will be illustrated by the system logic design description of the RCA 601 System in the following discussion. Specific elements have been selected from numerous possibilities to constitute the presently offered product line. The objective here is far from academic in that this approach provides, first, an exceptional ability for custom fitting of system elements for a specific user. Second, it tends to delay the inevitable onset of obsolescence by permitting revision of systems elements to increase performance or modify the orientation of the system according to the contemporary fashion in data processing.

These, of course, are supplementary to the common objective of commercially available systems, a cost-to-performance ratio in conformity with the vintage of the hardware. The host of sagacious decisions required in the selection of such appropriate components as transistors, packaging, and memory at optimum points of their cost vs. speed considerations, will not be elaborated but can be assumed by the reader.

A. THE RCA 601 SYSTEM

The RCA 601 System stresses a generalized system logic design in the computer. The main frame is a fast processing unit which includes a 1.5-microsecond memory. Its design features provisions for uniting other system elements into an integrated system by means of standard interfaces. A unique modular packaging concept is used for these system elements to allow efficient and flexible system combinations. A system diagram is shown in Figure 1. The all-solid-state elements of the computer system include a main frame processing unit, memory units, an arithmetic unit, and transfer channels. The main frame processing unit includes a 56-bit word memory module, a fast basic arithmetic unit, an input-output control, and a console transfer channel. Facilities for operating any combination of other elements comprising a single system are provided by three channels — the memory channel, the control channel, and the input-output channel.

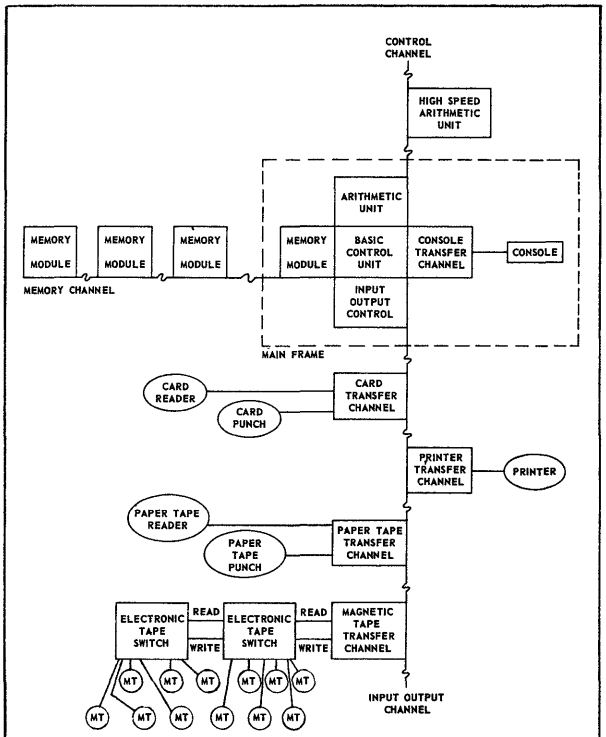


FIGURE 1. RCA 601 SYSTEM DIAGRAM

Additional memory modules via the memory channel are available to operate as asynchronous, independent units. In the RCA 601 System, asynchronism means that the occurrence of an event begins upon fulfillment of a set of machine status requirements, and terminates as soon as its own requirements are fulfilled. A high-speed arithmetic unit can be added via the control channel to comprise an expanded computer. This unit operates at higher speeds than the main frame arithmetic unit and performs full-word-binary and decimal-floating-point arithmetic.

Remaining elements consist of transfer channels which are links to peripheral devices via the input-output channel. Peripheral devices are available in card readers and punches, high-speed printers, paper tape readers and punches, and magnetic tapes. Flexible combinations of these devices within very broad constraints may be selected to operate on-line with the

system. Two sets of tape stations are native to the RCA 601 System — providing the choice of 100-kc. or 180-kc. decimal digit nominal transfer rates.

B. HARDWARE FEATURES

The generalized system logic design includes unique features such as elementary operation mechanization, an asynchronous control system, a generalized arithmetic unit transfer channel input-output control, and an asynchronous memory system.

A unique feature of the 601 system is the manner in which its instructions are implemented. Instructions are broken down into small component parts called "elementary operations". A sequence of elementary operations comstitutes the logical steps comprising a machine instruction; thus, the elementary operation corresponds to an instruction in much the same manner as an instruction corresponds to a routine. Instructions are performed by evoking these elementary operation sequences and executing them one at a time. Thus, by simply changing or adding elementary operation sequences, the instruction complement may be added to or changed. This open-ended design of instruction complement enables the customer to adopt new techniques in his computer operation as they are needed.

Asynchronism in operation timing is a convenient tool for modular variability in which a configuration of system elements is not fixed. The control system, arithmetic unit, and data transmission employ this technique. To illustrate this feature, let the successive elementary operations, "transfer" and "set", be considered. The "transfer" elementary operation calls for the transfer of the information from one specified register to another within the computer. The "set" elementary operation transfers data from a memory location into a specified register. As shown in an exaggerated form in Figure 2, the "transfer" operation involves steps 1, 2, 3, and 4. The asynchronous aspect of the control system is illustrated between steps 1 and 2 and the occurrence of step 4. The "transfer" elementary operation terminates just after the data transmission (step 3) has started so that the next operation may begin. Thus, steps 5 and 6, which are the set up and memory addressing for the next "set" operation, overlap with the relatively long transmission. The asynchronous aspect of data transmission via the data busses is indicated by steps 3, 6, and 8. There is a unique detection circuit on the busses to detect echoes from the receiver register and terminate the transfer by the generation of a terminate pulse, T. In this way, data transmission time depends on the physical configuration of the source and sinks involved. Note that steps 6 and 3 overlap in time because separate busses are involved; if the same bus is used then step 6 should be interlocked to prevent its initiation until the T signal is generated.

The RCA 601 System memory storage employs a word format, even though the data format is extremely flexible. The majority of data manipulation involves logic with certain arithmetic properties. An efficient and compact design is made by generalizing the basic arithmetic unit to include data handling functions as well as arithmetic operations. Operands that can be

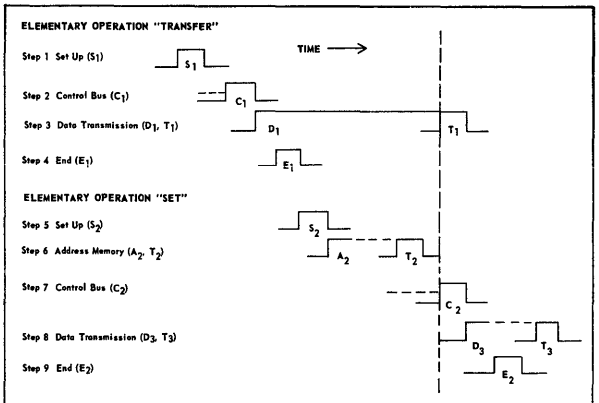


FIGURE 2. ILLUSTRATION OF ASYNCHRONOUS OPERATION AND OVERLAP

manipulated may be in character, half-word, or word format. Wired-in arithmetic operations include the full range of add,subtract,multiply, multiply and accumulate, and divide, for fixed-point decimal operands and for the majority of fixed-point binary operands. By the addition of a high-speed arithmetic unit, basic system speeds of certain operations are increased and floating-point arithmetic is added to the instruction complement. A further feature of the basic arithmetic unit design is its built-in ability to accommodate variable size characters. Four different character sizes, namely lengths of 3, 4, 6, and 8 bits, may be directly addressed and manipulated. A character length register is used to designate the size of character in all character operations. This register can be loaded and re-loaded with appropriate designators during a program by half-word set-register instructions. Operations other than character handling operations are not affected by this register. The variable size character handling ability together with the general symbol reoognition feature allows the RCA 601 System to handle a large variety of codes. A block diagram of the basic arithmetic unit is shown in Figure 3. The adder operation is an asynchronous circuit with respect to carry propagation. That is, as soon as the carry propagation subsides, the adder operation is terminated.

Peripheral devices in the RCA 601 System are handled by control buffer packages called transfer channels. These transfer channels work on-line with the input-output control of the main frame by an automatic instruction interrupt technique. The input-output channel is a general, standardized interface. Thus, looking out from the input-output control, proper operation is maintained, regardless of the number or the complement of the transfer channels in operation. This allows a generous flexibility in the make-up as well as the size of the input-output system. This transfer channel concept, coupled with a general, all-purpose input-output instruction complement, makes an efficient and economical way of coping with future development of peripheral and custom devices. The transfer channels are logically complete for independent and simultaneous operation with the main frame. Additions of transfer channels mean potential additions in amounts of simultaneity. The amount of simultaneity is automatically regulated by a built-in artificial, simple calculation called speed-weight. The transfer

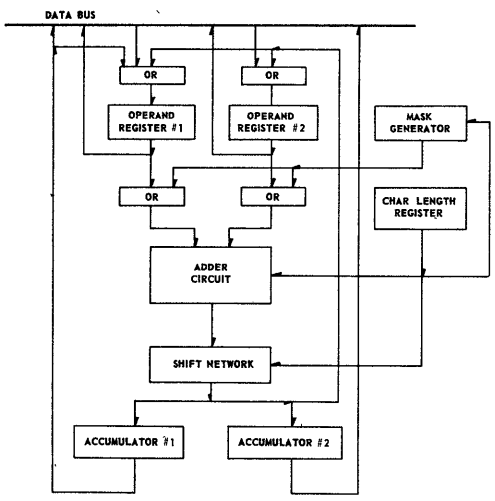


FIGURE 3. BASIC ARITHMETIC UNIT BLOCK DIAGRAM

channel concept also makes it possible for the operating console to operate interpretively, and communicate with the computer without having to stop operation.

A key feature in the RCA 601 System is the high-speed memory. Each 56-bit word is accessed in 0.9 microseconds and the complete address-read-write cycle is 1.5 microseconds. The memory operation is logically controlled and asynchronous. Separate commands to operate the three memory logical steps (address, read-out, and write-in) are directed from the control system. Thus, its operation is integrated with the control system. Overlapped operation is possible when more than one module is present. Generally three modes of operation can be accomplished:

1. Address and read-out.
2. Address and write-in.
3. Address, read-out and write-in.

The design approach here is basically conservative. The application of commercially existing components and proven techniques to solve the problem of this advanced equipment design have been used rather than depending on the development of new components. The design objective was to make the cost competitive with presently available memory systems that are up to eight times slower. The design stresses were on cost, reliability, and simplicity, taking into account the expected maintenance problem. The storage elements used are magnetic cores. The core size chosen has an I.D. of 18 mils; an O.D. of 30 mils, and a thickness of 10 mils. This small size is chosen for performance as well as for compactness of mechanical packaging. The circuit design simplifies the wiring complexity that is required in most magnetic core memories.

The product design of high-speed computers must deal with problems arising from multitudes of short pulses with extremely fast rise times. In order not to affect proper performance of these high-speed circuits, wiring techniques become complex. Wire lengths must be kept to a minimum. This is particularly true in the main frame processing unit. High transistor packing density is specified in step with the contemporary state of mechanical component packaging.

Despite preventative maintenance, computers occasionally become inoperative due to component or mechanical failure. Ready access directly to the internal construction means easier servicing with a minimum loss of costly computer time. A basic requirement achieved in the RCA 601 System package is that both the wiring side and the plug-in side may be opened for servicing without affecting the operating status of the computer. Means have been designed to make it easy to tap any point on this wiring for observation.

Transfer channels, additional memory units, and future expansion units are packaged in modules which are installed in universal racks. A unique channel-jumper technique is used. These channel-jumper cables thread through the entire system in a manner illustrated in Figure 1.

C. PROGRAMMING FEATURES

One vogue in the majority of recently-announced computing systems is the relative brevity of program statement contrasted to systems currently in common use. Factors contributing to these concise programs include powerful, well-integrated order codes, complex address-modification capability, flexible addressing schemes and variable instruction length.

Provision for variable length data has been made in early electronic machines and the similar motivations of efficient memory utilization and the elimination of "filler" material have extended this to include, also, those bits, in memory, which direct the control unit of the machine. In the RCA 601 System, "variable instruction length" means that instructions may be either 1, 2, 3, or 4 half-words in length.[1] A unit data length in the RCA 601 System is either a word (or half-word) or a character. One of these, the half-word of 24 information bits, is the unit length for instructions.

Generally an instruction half-word is an operation half-word that may have as many as three address half-words appended to it. A distinction has to be made between the number of addresses contained in an instruction and the number of addresses utilized as an integral part of the execution of an instruction. Each instruction type utilizes a fixed number of these address registers, e.g., a MOVE will utilize two address registers, MULTIPLY utilizes three, a DO instruction utilizes none at all, etc., and frequently those are the number of addresses written in that instruction. However, the address registers utilized by an instruction have their contents augmented by one data-length unit. This value of the address register is often the appropriate value of the register for the operation of the next instruction. In particular, this occurs when the

[1] Input-output instructions are the exception to this statement.

data of interest is stored in contiguous arrays in memory. In such a case, when an address register contains the desired value, then that address need not be written in the program statement. Three bits termed "assumed bits" in the operation half-word of an instruction specify whether or not address half-words for each of three address registers are written following the operation half-word. Thus, a lesser or a greater number of addresses than used by the instruction may be written. When fewer addresses are used, the remaining addresses are said to be "assumed"; when additional addresses are used, they merely cause the address registers to become loaded. The stepping of the address registers in conjunction with assumed addressing is really equivalent to automatic address modification by unity, with, however, several advantages over address modifiers: no address modifiers are used up for this purpose; no time is lost to accomplish the address modification; no space is required in the program for the address; and no time is required to access the address.

Another place in which explicit addresses are not written occurs when the accumulator is being used in arithmetic operations. Again three bits in the operation half-word specify whether any or all of the operands refer to the contents of the accumulator. Thus, a single half-word suffices to instruct the accumulator to double itself, an ADD instruction with all three addresses assumed, and the accumulator specified as each of the operands. Figure 4 illustrates some of these features.

A unique and substantial addressing flexibility is incorporated in the address half-word. The 24-bits of this half-word, shown in Figure 5, are divided as follows: an address part of 19 bits; 15 for addressing 2^{15} words; a 16th to address half-words; and three additional bits to address characters within the half-word. The 20th bit is used to designate indirect addressing. The four remaining bits are used to address any of eight address modifiers.

		OPERATION	ADDRESS A	ADDRESS B	ADDRESS C
EX. 1: $\sum_{i=1}^{n} a_i$	DO n	B. ASSUMED			
		C. ASSUMED	a_n	....	....
IN CONSECUTIVE WORDS					
	ADD	A. ASSUMED	....	....	....
		B. ACCUMULATOR			
		C ACCUMULATOR			
TOTAL PROGRAM: THREE HALF-WORDS					
EX. 2: $\sum_{i=1}^{n} a_i b_i$	DO n	C. ASSUMED	a_n	b_n	....
	MULT/ACC	A. ASSUMED	....	....	....
		B. ASSUMED			
		C. ACCUMULATOR			
TOTAL PROGRAM: FOUR HALF-WORDS					
EX. 3: a^n	DO n	A. ASSUMED	....	....	....
		B. ASSUMED			
		C. ASSUMED			
	MULTIPLY	B. ACCUMULATOR	a	....	....
		C. ACCUMULATOR			
TOTAL PROGRAM: THREE HALF-WORDS					

FIGURE 4. ASSUMED ADDRESSING AND STEPPING OF ADDRESS REGISTERS

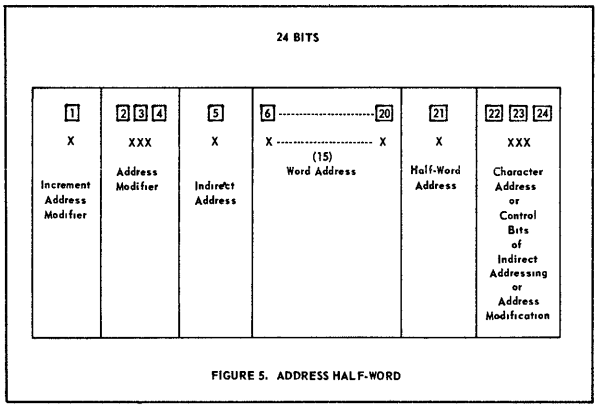


FIGURE 5. ADDRESS HALF-WORD

Seven of the address modifiers are a full word in length, where the left half-word is the value part applied to an address when selected, and the right half-word is an increment part selectively added to the value part. This selectivity is gained by providing two addresses for each address modifier, one of which causes an automatic incrementing to take place. This accounts for the generous use of four bits to select eight address modifiers.

Applying both address modification and indirect addressing to an address would normally require an arbitrary order of precedence. However, in the RCA 601 System whenever an indirect address is specified, it selects another half-word in which is contained another address. Therefore, the three bits used for character addressing in this case, might be superfluous, but are used instead for the following:

1. To defer application of the address modifier with the indirect address to the direct address. This can permit dual address modification on the direct address.

2. To inhibit address modification on the direct level.

3. To inhibit subsequent indirect addressing.

Figure 6 illustrates the possibilities with this octal digit for the case of only a single level of indirect addressing.

Indirect addressing and the flexible application of address modifiers provide considerable facility in working with address lists. Lists of addresses can be generated by magnetic tape read instructions which list the addresses of special symbols (designated by the programmer) in memory, as well as the information itself. It is then possible to eliminate much data movement by utilizing the address lists.

Efficient data encoding is carried a step further in the character handling capability of the RCA 601 System. It has been mentioned that four different character sizes (lengths of 3, 4, 6 and 8 bits) may be directly addressed and manipulated. Thus, sequences of decimal digits are normally handled in four-bit characters, alphanumeric data in six bit characters, etc., providing efficient tape and memory storage.

BITS OF ADDRESS HALF-WORD		4	1	16	3
ADDRESS A:		AM1	I	B	d
ADDRESS B:		AM2	–	C	–
ADDRESS B+(AM1):		AM3	–	D	–
d PERMITS:					
INDIRECT ADDRESS TO BE		1) B			
	OR	2) B+ (AM1)			
DIRECT ADDRESS TO BE		1) C+ (AM2)			
		2) C+ (AM1)			
		3) C+ (AM1) + (AM2)			
	OR	1) D+ (AM3)			
		2) D+ (AM1)			
		3) D+ (AM1) + (AM3)			

FIGURE 6. CONTROL OF INDIRECT ADDRESSING AND ADDRESS MODIFICATION

The RCA 601 System features sets of indicators to provide a parallel decision making ability concurrently with the running program. Conditions sensed include: arithmetic underflow and overflow; error conditions; unsuccessful scanning operation; result positive, negative or zero; and a binary indicator specifying such information as whether a logical connective yielded zero. A programmer-set mask is associated with each indicator to permit or inhibit an automatic branch whenever the condition is encountered. This feature permits minimizing the length of repetitive loops by eliminating, each time, explicit sensing for rare conditions.

When one of these automatic jumps does occur, storing of relevant registers takes place automatically, assuring ability to restore the status of the machine whenever control is returned to the interrupted point. In particular, one of the conditions which will cause an automatic jump to occur is the termination of an input-output operation. The format of an input-output instruction differs from the rest of the order code, including up to 5 address half-words. One of these address half-words contains an address to which control is to be transferred when the operation is complete. This permits an effective multi-programming scheme, where linkages between programs occur such as to permit maintenance of input-output rates.

Figures 7 and 8 present a brief description of the characteristics and representative times of the current RCA 603 Computer in the RCA 601 System.

MEMORY MODULE SIZE	8192 WORDS, 56 BITS EACH UP TO 4 MODULES	
OPERATION TIMES:		MICROSECONDS
A + B → C, 1-WORD		6.6
ACCUMULATOR + B → ACCUMULATOR, 1-WORD DECIMAL		3.2
A × B → C, 1-WORD DECIMAL (NO ZEROS)		69.4
A → B, 10 WORDS		32.0
A → B, 6 6-BIT CHARACTERS		15.1
A → B, 6 4-BIT CHARACTERS		11.3
BRANCH		0.5
DO		1.5 to 3.0*
INSTRUCTION ACCESS AND INTERPRETATION TIMES		
ACCESS		2.5 to 7.0*
INDIRECT ADDRESSING		1.7
INDEXING		2.9 to 3.5*
INPUT-OUTPUT CONTROLS		
UP TO 16 SIMULTANEOUS INPUT-OUTPUT OPERATIONS		

* DEPENDS ON PROGRAM SEQUENCE

FIGURE 7. MAIN FRAME COMPUTER

CARD:	
READ	600 CARDS PER MINUTE
PUNCH	100 CARDS PER MINUTE
PAPER TAPE	
READ	1000 CHARACTERS PER SECOND
PUNCH	300 CHARACTERS PER SECOND
PRINTER	
120 CHARACTERS PER LINE	600 LINES PER MINUTE
MAGNETIC TAPE	
	180-KC DECIMAL DIGIT RATE
	100-KC DECIMAL DIGIT RATE
	50-KC DECIMAL DIGIT RATE

FIGURE 8. PERIPHERAL DEVICES

ASSOCIATIVE SELF-SORTING MEMORY

Robert R. Seeber, Jr.

Product Development Laboratory, Data Systems Division
International Business Machines Corporation
Poughkeepsie, New York

Summary

A cryogenic associative memory is proposed in which the status of a memory word is determined, with respect to an interrogating word, on a high, low or equal basis. Thus the bracketing pair of words is determined, allowing the interrogating word to be inserted between them, a "dummy" register holding it temporarily. A double-shuffle operation moves the other words to make room for the new word in proper sequence.

Introduction

One of the major problems that occurs in the use of a computer, particularly in business uses, lies in the sorting of data. The importance of this problem can be judged on the basis of the hundreds of pages of description of sorting routines that have been written. Programs are in some cases many thousands of instructions in length. Heretofore, the attack on the sorting problem has been almost entirely on the programming level. In this paper is proposed a memory system organization to achieve sorting within the memory. By the use of associative memory principles, extended to facilitate sorting, a new approach to sorting is developed. Words to be sorted enter memory in random order but each is placed within memory in its proper relationship to previously sorted words.

Associative Memory

Associative memory may be defined as memory in which a word of data is retrieved on the basis of part or all of the data content of the word. Words are stored in vacant registers and subsequently are recovered not by naming the location of a register but rather by naming a portion of the word as an identifier. This identifier or tag may in some cases be a portion added to the data word for the identification purpose. However, in the more general case, which we call a fully associative memory, the data word can be retrieved by using selected portions of the word itself as its identifier; in this case, the remaining portions are masked out. Thus the knowledge of the exact storage location in the physical sense may be completely immaterial to the operation. A more detailed explanation of a simple associative memory was given in an earlier paper.[1]

Sorting with an Associative Memory

The use of an associative memory reduces somewhat the need for sorting, particularly in the case where sorting is used to provide means for locating particular words by their ordered identifier or argument, as is usual in table look-up operations. Since an associative memory allows direct access by these identifiers, sorting for these purposes is not necessary. But for other purposes, particularly for sorting output lists, sorting is required. This may be done by an associative memory by arranging to retrieve in order all possible combinations of the given identifier. For example, if we have a 3-decimal digit identifier and want to retrieve 965 different data words identified by this 3-digit code, we can set up a counter to provide us with all combinations running from 000 through 999 to act as identifiers for the 965 words. The efficiency will be quite high, requiring 1000 retrieval tries to produce the actual 965 retrievals in order. However, this system breaks down in the more frequent case of the identifier not so densely coded. For example, a 10-decimal digit part number code may have only a few thousand different parts which would require 10^{10} retrieval tries to recover in order. Since sparsely populated codes seem to be the more general rule, particularly in business problems, a means for actually sorting the words would be very desirable.

Proposed System

In a general associative memory a simultaneous comparison is made between an entry word and all of the word registers in the memory to determine the match or non-match status of each word. This is done by providing comparison circuits between the entry register and each of the word registers such that an equal or unequal status is determined. This has previously been done for the purpose of retrieving a matched word from memory. For

writing into memory in order, we now add the requirement that these comparison circuits be extended to provide a comparison indication on high, low or equal for each of the word registers. This then will supply us with enough information so that we can determine where within a previously sorted sequence a new word should be inserted. Additional registers called dummy registers are supplied between each of the word registers. This allows an incoming word to be placed between the proper two words in memory; then, by a double-shuffle transfer cycle, the words in memory are shuffled to enter the new word into its proper place.

Figure 1 is a block diagram showing this arrangement. At the top there is an entry register where data words are coming in from some other portion of the computing system. Comparison circuits from this entry register extend through all of the word registers and each of the word registers supplies indication as to whether it is low, high or equal to the word in the entry register. From this information, the dummy register lying between the two word registers which bracket the word in the entry register can be selected and the word then can be transferred from the entry register to that dummy register. On this same half cycle, all the words in word registers above the selected dummy register move up to the respective dummy registers immediately above each of those word registers. On the next half cycle, while a new word is entering the entry register, the words in dummy registers move up to word registers immediately above those dummy registers, thus leaving the words in memory again in word registers in proper order including that word just entered.

Means are provided for exiting one word from dummy register 1, that is, the forward exit register, as the memory fills up. That is, as the memory fills up, earlier words are pushed out the top. In this case, the echo register retains the information of the word just exited.

Another mode of operation is provided so that words coming in through the entry register may be accepted in inversely sorted order. In this case, as the registers fill up, words are pushed out the bottom through the backward exit register; this inverse function may be useful in handling partly sorted blocks of words read backwards from a tape on which data has first been stored in the normal forward order. This operation may reduce the need for re-winding operations when tapes are used in conjunction with this sorting memory.

Sorting Example

The chart of Figure 2 shows the successive cycles of the operation of sorting 21 different words of which the sorting identifiers are shown on the entry line. It is assumed that the memory for this example has 5 word registers, W1 through W5, and 6 dummy registers, D1 through D6. The two-digit identifier has a bar over it when the word has just arrived at the location so indicated. A code with no bar indicates that the word is still present at that location but has been moved elsewhere, i.e., this is a "shadow" word. Where a bar appears under an identifier, this indicates that the word has not been moved on that cycle but has been previously moved to that location.

Thus, it may be seen in Figure 2 that the number 29 is placed in the entry register during the A portion of cycle 1. During the B portion of cycle 1, the number 29 is transferred to dummy register W + 1. During the A part of cycle 2, the number 1 is placed in the entry register and the number 29 is transferred from dummy register W + 1 into word register W, where in this instance of illustration, W equals 5. During the B part of the second cycle, the 1 is transferred into dummy register W where W is equal to 5 in this instance, and the number 29 is transferred from dummy register 6 into word register 5. Since the number 1 in the entry register was less than the number 29, it will be observed that the number 1 was placed nearer the top of the column of registers. During the A portion of cycle 3, the number 44 is placed in the entry register, the number 1 is transferred from dummy register 5 to word register 4, and the number 29 remains in word register 5, the image being in dummy register 6. During the B portion of cycle 3, the number 44 is moved into dummy register 6 replacing the image 29, the number 29 is moved into dummy register 5 and the number 1 is moved from word register 4 into dummy register 4. The first three cycles have illustrated the cases where the number in the entry register was less than the numbers stored in memory and the other instance where the word in the entry register was greater than those stored in memory. The next example of cycle 4 concerns merging a number among those previously stored.

In the A part of cycle 4, the number 10 is placed in the entry register, 1 is moved to word register 3, 29 is moved to word register 4, and 44 is moved to word register 5, thus

making available the vacated dummy registers. During the B part of cycle 4, the number 10 is inserted in dummy register 4 and the number 1 is moved from word register 3 into dummy register 3. In this manner, a word is merged with those in memory.

Cycle 7 illustrates the overflow of a word to the output bus; cycle 11 illustrates the start of another block of words at the end of memory.

Cryotron Circuits

When cryotron circuits are available, they would appear to have ideal properties for such a memory. It should be noted that in the self-sorting memory, as in other associative memories, there is a great deal of distributed logic. The cryotron is a single element which can be used for both storage and logical purposes. An implementation of the self-sorting memory in single-crossing, thin-film cryotrons is proposed. Cryotron circuits have previously been described by Dudley Buck.[2] Here we use a simplified symbolism, a gate being shown by a semi-circle with its diameter lying along the gate line and the corresponding control wire at right angles to the gate wire and bisecting the semi-circle. Figure 3 is the configuration for a flip-flop with read-in and read-out circuits employing this symbol.

The top current source (denoted by a "+") splits into two paths, only one of which is superconductive at a time. If the left path is conducting, the flip-flop is said to be "on" or contain a "1"; if the right path is conducting, the flip-flop is "off" or contains a "0." The feedback action of the flip-flop is accomplished by the top or bottom cryotron, depending on whether the flip-flop contains a "0" or a "1." If it contains a "1," the left path of the flip-flop is conducting. This current through the lower cryotron control makes the right path resistive and keeps the current flowing in the left path. Similar action takes place at the top cryotron when the right-hand path is conducting.

Assume that we have a "0" in the flip-flop, and we want to change its state, that is, read in a "1." We cause a current flow through the control path of the "read-in 1" cryotron, making that cryotron resistive. Since both the right and left paths of the flip-flop are now resistive, the current divides in half. When the current through each path falls near the half point, neither the upper nor the lower feedback cryotron is resistive. This leaves the "read-in 1" cryotron as the only resistive element, forcing all the current to flow through the left-hand path and making the lower cryotron resistive. The flip-flop has now reached a stable state in the "1" or "on" condition, and the "read-in 1" current can be removed. Similarly, we can change back to an "off" condition by applying a current through the "read-in 0" cryotron.

The read-out is accomplished by completing a circuit from the read-out current source through one of the read-out cryotrons and to the output device. If we assume that the flip-flop is in the "0" state, current will be flowing through the right path and through the control path of the "read-out 1" cryotron, making that cryotron resistive. Current then flows through the superconducting "read-out 0" cryotron gate to the output device, where a "0" will be sensed. The circuit is similar through the "read-out 1" cryotron when the flip-flop contains a "1."

Cryotron Bit Position

The heart of the memory system lies in the data bit position shown in Figure 4. In this figure a portion of a data bit for word register W-1 is shown near the top of the figure and the complementary portion for word register W is shown near the bottom of the figure. In between is the flip-flop for the dummy register lying between these two word registers. At the top of the drawing there are provided read-out and read-in circuits for going from or to that word register to or from the dummy register immediately below. In the dummy register flip-flops, there are shown corresponding read-in and read-out circuits for transfer between the dummy register and the word register above or below it. There is also shown the select circuit which provides for the storing in this dummy register of a data word coming from the entry register.

For word register W there are shown entry and exit transfer circuits for coming from or going to the dummy register above it. Next is a match circuit which controls the exit of data from this word register when it is a matched register for the read-out of data in the normal associative manner. The last lines on the figure show the equal, low and high matching circuit for determining the selection of the proper dummy register.

The vertical lines extending through memory are the entry and exit busses for the "0" (on the left of the bit) and for the "1" (on the right of the bit);also the "0" and "1" compare lines (on either side of the storage loops), carrying the information from the entry register to be compared with the bit status. The no compare line will carry current instead of either the "0" of "1" line if this bit position is to be masked out, thus forcing an equal comparison as far as this bit is concerned.

As an example of operation, assume that the bit shown is the right-hand bit of the interrogating tag in a forward sorting operation. Bits to the left have shown an equal status. This bit is a "1" compared with a "0" in the entry register. Current flowing into the comparing circuit of this bit on the equal line, from the left, will be blocked from the three upper comparing lines and permitted to flow through the bottom line, thus onto the high line into the control section of this word, as shown in Figure 5. Assuming that this word, W, is the first word in memory having the "high" status, the control circuits will operate the dummy out line during the "A" half of cycle, thus moving the word from dummy register W up to word register (W-1) completing the previous sort operation. During the second half cycle, "B," the word from the entry register will be entered in dummy register W preparatory to being moved into word register (W-1) on the "A" half cycle of the next sort operation.

Figure 5 shows the circuits for operating the entry and transfer circuits as controlled by the high, low, and equal signals.

Conclusions

By extending the properties of an associative memory, it is possible to secure a self-sorting memory. The work thus far done is of a purely systems nature since the proper components are not yet available. With a self-sorting memory, programming can be materially simplified and the internal sorting procedure improved.

Acknowledgments

The assistance of Arthur J. Scriver, Jr. is gratefully acknowledged.

References

1. Cryogenic Associative Memory by Robert R. Seeber, Jr., National Conference of the Association for Computing Machinery, Milwaukee, Wisconsin, August, 1960.

2. The Cryotron -- A Superconductive Computer Component by D. A. Buck. Proceedings of the IRE, April, 1956.

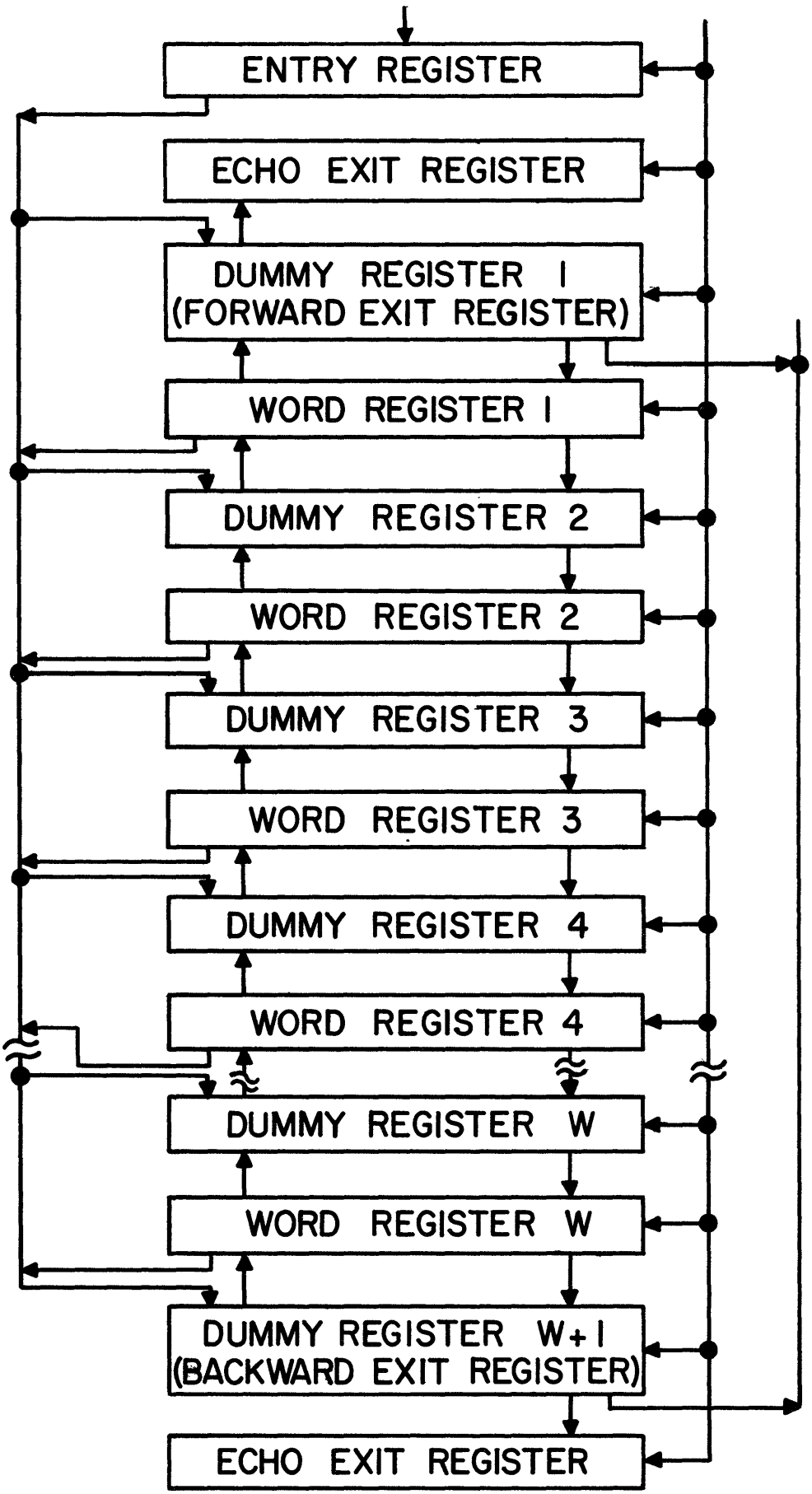


Fig. 1. Memory System.

CYCLE	1		2		3		4		5		6		7		8		9		10		11		12		13		14		15		16		17		18		19		20		21	
PART	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B
ENTRY	29	29	1	1	44	44	10	10	11	11	51	51	39	39	36	36	12	12	50	50	23	23	55	55	22	22	58	58	42	42	64	64	82	82	25	25	98	98	31	31	38	38
ECHO													1	1	10	10	11	11	12	12	29	29	36	36	39	39	44	44	50	50	51	51	55	55	58	58	64	64	82	82	98	98
EXIT (DI)												1	1	10	10	11	11	12	12	29	29	36	36	39	39	44	44	50	50	51	51	55	55	58	58	64	64	82	82	98	98	22
W1											1	1	10	10	11	11	29	29	29	29	36	36	39	39	44	44	50	50	51	51	55	55	58	58	64	64	82	82	98	98	22	22
D2										1	1	10	10	11	11	29	29	29	29	36	36	39	39	44	44	50	50	51	51	55	55	58	58	64	64	82	82	98	98	22	22	23
W2									1	1	10	10	11	11	29	29	36	36	36	36	39	39	44	44	50	50	51	51	55	55	58	58	64	64	82	82	22	22	22	22	23	23
D3								1	1	10	10	11	11	29	29	36	36	36	36	39	39	44	44	50	50	51	51	55	55	58	58	64	64	82	82	22	22	22	22	23	23	25
W3							1	1	10	10	11	11	29	29	39	39	39	39	39	39	44	44	50	50	51	51	55	55	58	58	22	22	22	22	22	22	23	23	23	23	25	25
D4						1	1	10	10	11	11	29	29	39	39	39	39	39	39	44	44	50	50	51	51	55	55	58	58	22	22	22	22	22	22	23	23	23	23	25	25	31
W4					1	1	29	29	29	29	29	29	44	44	44	44	44	44	44	44	50	50	51	51	55	55	22	22	22	22	23	23	23	23	23	23	25	25	25	25	31	31
D5				1	1	29	29	29	29	29	29	44	44	44	44	44	44	44	44	50	50	51	51	55	55	22	22	22	22	23	23	23	23	23	23	25	25	25	25	31	31	38
W5			29	29	29	29	44	44	44	44	44	44	51	51	51	51	51	51	51	51	51	51	23	23	23	23	23	23	23	23	42	42	42	42	42	42	42	42	42	42	42	42
D6		29	29	29	29	44	44	44	44	44	44	51	51	51	51	51	51	51	51	51	51	23	23	23	23	23	23	23	23	42	42	42	42	42	42	42	42	42	42	42	42	42	42

Fig. 2. Sorting Example.

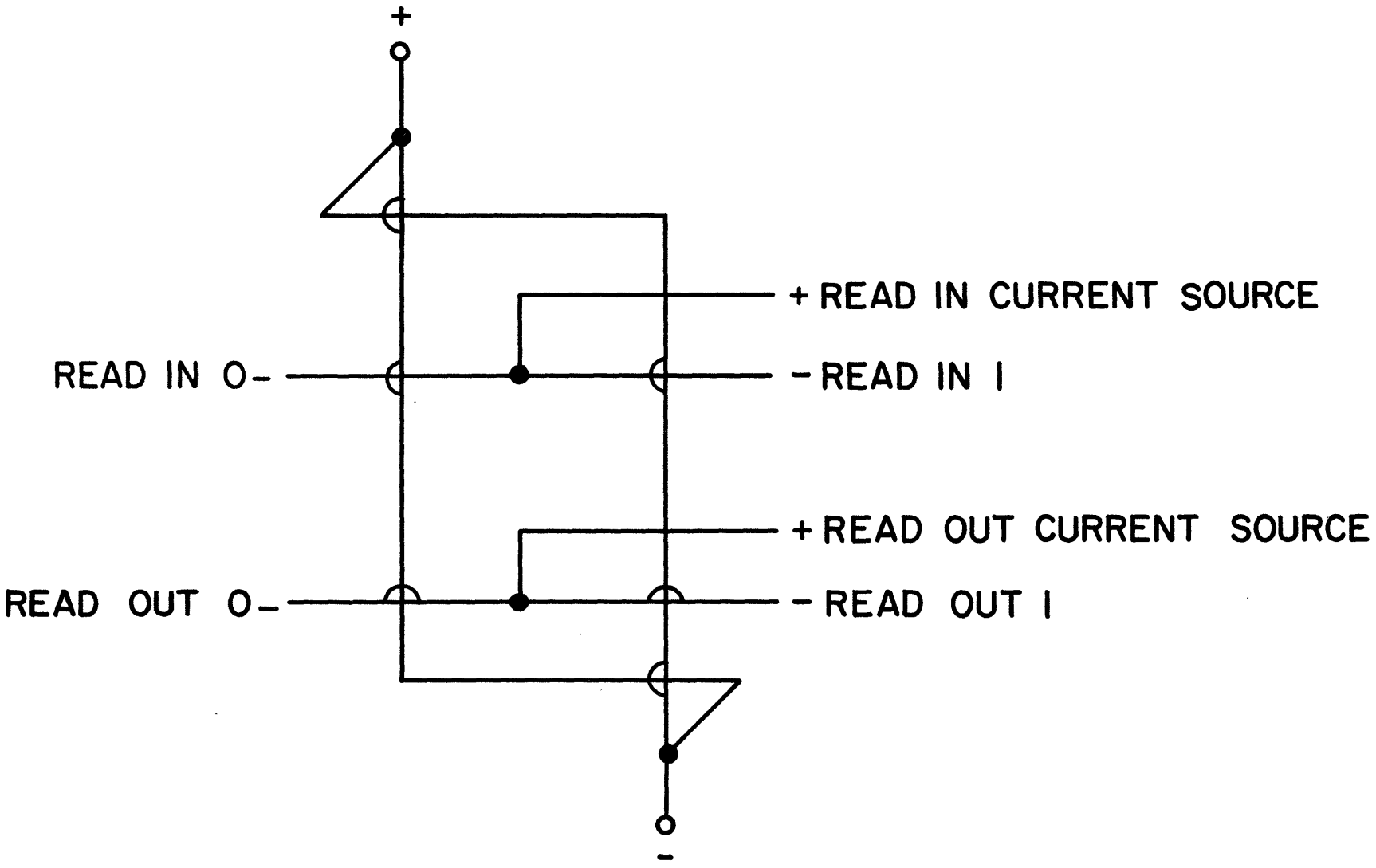


Fig. 3. Cryotron Flip-Flop.

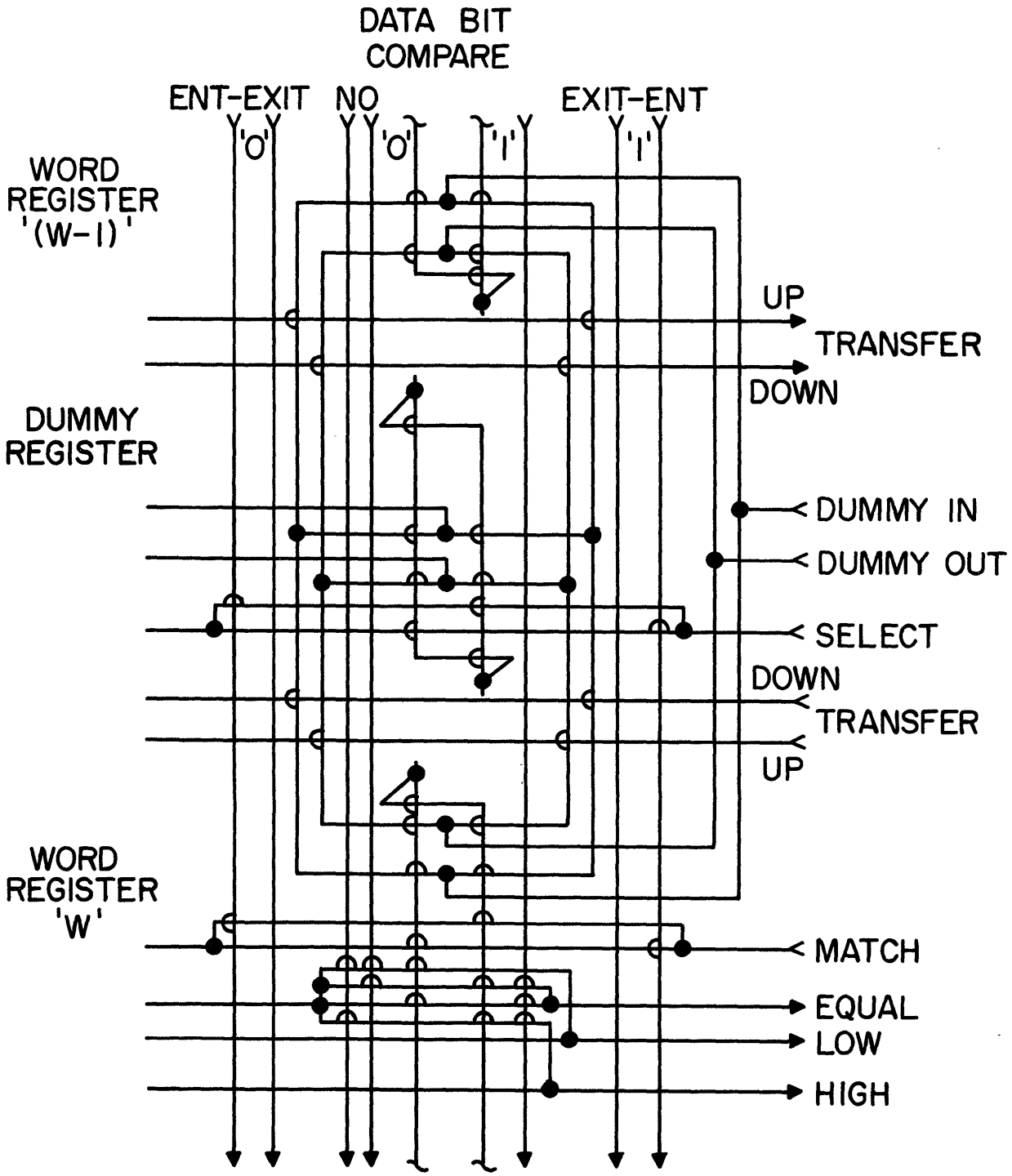


Fig. 4. Data Bit.

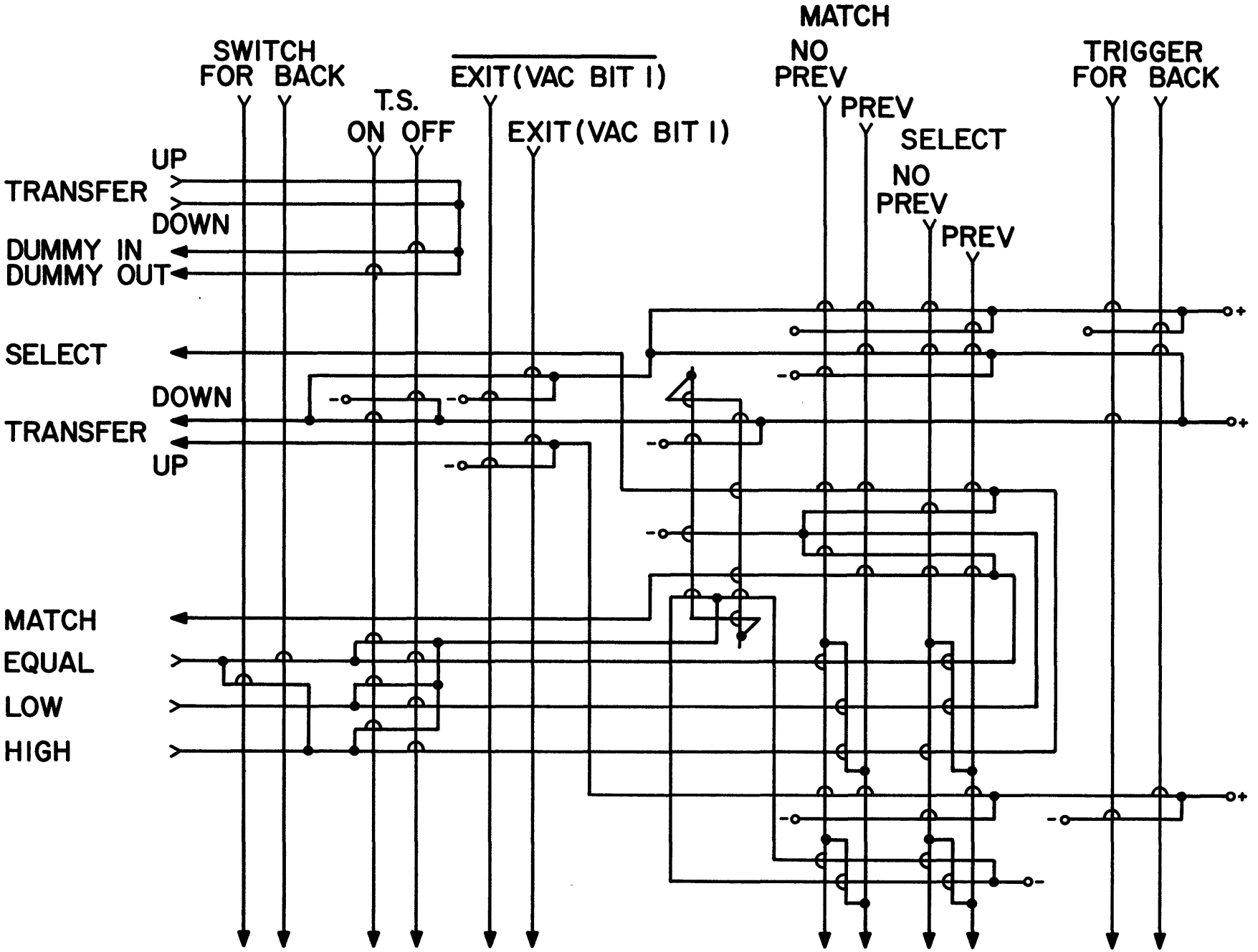


Fig. 5. Word Register Control.

UNIVAC® - RANDEX*II
RANDOM ACCESS DATA STORAGE SYSTEM

G. J. Axel

Remington Rand Univac
Division of Sperry Rand Corporation
Philadelphia, Pa.

SUMMARY

A random access drum file, having 198.6 million bits total storage capacity, a bit density of 650 pulses per inch, and 385 milliseconds average data access time, is described in this paper.

Two flying-magnetic-recording-heads transfer data to and from the drum file unit. They are self-supported, by a hydrodynamically generated air film, over two magnetically plated drums (24 inches diameter and 44 inches long). The heads, drums, and head-positioning servo are enclosed in a sealed and pressurized chamber to prevent their contamination by foreign material normally found in the atmospheric air.

The text describes in detail:

(1) The logic of operation and description of the overall drum file.

(2) Construction of the flying-heads.

(3) Descriptions of the servo and mechanical adder, which position the flying-heads over selected addresses on the drums.

INTRODUCTION

RANDEX*II, developed by Remington-Rand Univac, is a mass storage, random access drum file (See Figure 1), capable of being used with any one of a number of UNIVAC computing systems, both present and future. It is presently designed for use with a biquinary (5, 4, 2, 1) computing code, however, other codes can be readily incorporated. Initial use of this drum file will be on the UNIVAC®- Solid State Computer; ten drum files can be used on this computer in conjunction with a single drum control unit.

The drum control: 1) decodes computer instructions for the drum file systems and issues the commands to the drum file(s) to process these instructions; 2) supplies d-c voltages to the drum file system circuits; 3) provides the data transfer medium between drum file(s) and computer; and 4) functions as an off-line control for the mass storage drum file system.

* Trademark of the Sperry-Rand Corporation.

Historically, the RANDEX II drum file follows the development of the UNIVAC - LARC[1] and RANDEX I drum files. All three units use the hydrodynamically supported flying-magnetic-recording-head principle, and magnetic plated drums 24 inches in diameter. Whereas the LARC drum file contains a drum 28 inches long, one sequential head-positioning mechanism and a single flying-head, the RANDEX drum file contains two drums 44 inches long, one random access head-positioning mechanism and an individually flying-head for each drum.

The RANDEX I drum file was designed using very conservative bit and track densities to assure reliability of the initial design. Subsequent improvements in the head-positioning mechanism, finishing and magnetic plating of the large drums, magnetic recording elements of the flying-heads, and recording techniques brought about the development of the larger capacity RANDEX II. Specifications for RANDEX II are listed in the table of Figure 2.

DETAILED DESCRIPTION

The central element of the RANDEX drum file is a pressurized dust free enclosure (See Figure 3), which contains the following; two individually driven magnetic plated drums mounted on self-aligning roller bearings, and on which information is stored; the flying-heads used for reading and writing; and a head-positioning servo. A mechanical lever adder (discussed in detail below), located on the lower right section of the drum file, is used for additional head positioning, and is connected to the servo by a mechanical linkage. Above the dust free enclosure is an electronic deck on which the RANDEX control panel is mounted, and within which are located plug-in chassis and a transistor circuit card library (See Figure 4).

The dust free enclosure acts to keep foreign material away from the drums, flying-heads, and servo components. The enclosure is pressurized by blower action with incoming air, which is first filtered through a low-cost commerical filter to remove the largest percentage of foreign material, and then sent through a special filter which removes particles as small as 0.3 micron (approximately 0.00001 inches) with 99.97 per cent efficiency. Air is continuously circulated within the enclosure, passed over the servo motor for cooling, and then

re-circulated back through the above mentioned special filter by another blower, to assure permanent cleanliness. A window, which is part of the sealed enclosure, allows the operation of the drums, flying-heads, and servo to be observed from the front of the drum file. Doors in the rear section of the enclosure (See Figure 4) are located behind each of the drums to permit access to the drums recording surface for cleaning and inspection purposes. Easily removable end-bells on both sides of the enclosure (See Figures 3 and 5) permit access to the servo components and flying-heads. Contamination of the clean air within the enclosure is practically eliminated by permitting only small access areas to exist when inspection or maintenance of servo components and the flying-heads is desired. Clean areas are not required for these functions. A further guarantee of cleanliness when the end-bells are removed is assured by the fact that air flow is always out from the enclosure, due to internal pressurization.

Each drum consists of cast bronze end supports and a center support, upon which a 3/16 inch thick brass tube is shrunk. This assembly is fixed to one end of a supporting steel shaft and is permitted to float on the other end, thereby allowing for differential expansion resulting from temperature changes during assembly and operation. The drum is then machined on a lathe to provide a concentric, smooth, and uniform surface over which the flying-head will operate. Final steps in drum manufacture are Ni-Co magnetic plating, balancing, and testing for both head flying characteristics and plating quality.

Reading and writing of information on each of these drums is performed by an individual flying-head, which is supported over the drum surface by a hydrodynamically generated air film created by the rotating recording surface of the drum. No external means are used to provide an air film between the head and drum thereby avoiding unnecessary biasing of the head and possible localized contamination effects of the head and recording surfaces. Each head is gimbal-mounted and has three degrees of freedom, so that it can follow the drum surface and maintain a uniform head-to-recording surface spacing. Upon command a head lowering and raising mechanism lowers the head into flying position over the drum recording surface, without physical contact between head and drum surfaces. An area is set aside on one end of the drum for this function. Should it be desired to raise the head or stop the drums, or should some malfunction occur which may cause the head to touch or approach the drum recording surface closer than its established design spacing, the head raising mechanism will automatically lift the head away from the drum, regardless of location along the drum axis.

The two flying-heads are mounted on a common carriage (See Figures 5 and 6) that moves along a rail (located between the two drums) parallel to the recording surfaces. The carriage is driven by a cable that passes over a capstan on the shaft of a servomotor located in the left end-bell (Figure 5). A servo-potentiometer, located in the right end-bell (Figure 3) is connected to the carriage by a thin steel band. The potentiometer is a part of a bridge network in the servo positioning logic and is the carriage position indicator. Mounted parallel to the carriage rail is a movable notched rack (See Figure 6), connected to the lever adder on the right and biased against the lever adder by a spring on the left. Anyone of its 20 positions is determined by the output of the lever adder. There are 100 accurately machined and uniformly spaced teeth on the rack, whose shape can be seen in Figure 6. When the carriage is in position, over a preselected notch of the rack, as determined by the servo-potentiometer, a pawl is extended from the carriage and moved against the selected tooth. Therefore, by means of the servo and the lever adder, the carriage can be driven to anyone of 2000 discrete positions, called tracks, along the drum's recording surfaces. The combination of two drums and two heads then provides 4000 tracks of data per drum file.

Servo and lever adder movements are performed in parallel. Going to a new address may involve servo and/or lever adder changes. When a servo change is indicated, the carriage is withdrawn from the tooth, the pawl is retracted from the notch, and the carriage is moved to the newly selected track where the positioning cycle is repeated. The lever adder moves the notched rack to a new position, independent of the servo change. However, servo positioning circuitry must compensate for a lever adder movement. The repositioning of either or both of these mechanisms results in a new track address.

The electronic deck assembly contains a hinged transistor card library and four plug-in chassis; namely, servo-amplifier, servo-amplifier power supply, address register, and control chassis (See Figure 4). The transistor card library houses plug-in cards, which contain transistor circuits that are associated with flying-head positioning logic, read-write logic, and malfunction logic. The servo amplifier drives the control field of the servomotor; the servo-amplifier power supply supplies the d-c voltage required for the operation of the servo-amplifier.

The address register contains a relay network, which functions as a combined digital to analog converter and memory. It is used to decode and store the track address as it is received from the drum control unit. The control chassis contains relays associated with flying-head malfunction circuits, alarm circuits and other logic circuits.

The RANDEX control panel, mounted on the electronic deck assembly (See Figure 3), has

two sets of controls and indicators; one is for the use of the operator, and the other for use in maintenance. The operator's controls and indicators are located on the left front side of the panel; while special maintenance controls and indicators are located on the right front side of the panel, behind a normally closed access door in the casework (See Figures 1 and 3). All controls to manually operate the drum file off-line are provided on the operator's and maintenance panels, although specific changes in track addressing, supply voltages and read-write instructions must be obtained from the drum control. The carriage may be moved over the length of the drum, to approximate track positions, by use of a sweep control on the maintenance panel. A break-in plug is provided on the electronic deck to set new address selections in the address register relays from an external "black-box", if so desired, although this is not considered standard equipment.

Access doors are provided at strategic locations in the casework to permit easy access to the transistor card library, plug-in chassis, drum access doors, main power control box, and maintenance panel, without having to remove entire panels.

LOGIC OF OPERATION

The starting of a multiple number of drum files, associated with a particular drum control unit, is sequenced, so that only one drum file draws starting current at a time. The start command is sent to the first drum file through the drum control unit. When that unit has started, it supplies a start command to the next drum file. This starting sequence continues until all drum files have been started. Should any of the drum files be turned off at its control panel or switched to local operation by one of the maintenance controls, the start command is automatically bypassed to the next drum file. A drum file set in this condition is labeled as off-normal and will not accept commands issued from the drum control unit or computer.

As the drums in a file reach operating speed, voltages are supplied, in a preselected order, to the various control circuits and logic elements. Automatic clearing of all circuits and setting of logic elements as required prepares the machine for operation. The flying-heads carriage is automatically positioned over a prescribed area, toward the left end of the drums, where the heads are lowered to the flying position. Once heads are in the flying state, the carriage automatically moves to the address stored in the address register; if no address exists in the register, the carriage moves to track address 0000.

If all of the maintenance switches are in the normal position, and if no other off-normal condition exist, the control lines from the drum control unit are automatically connected to the drum file, and the file is then under command of the control unit.

Instructions from the computer can now be processed by the drum file system, or the drum control unit can be operated from its control panel for off-line operation of the drum file system. Since each drum file has its own memory for track address storage, it is possible to read or write on one drum file, while the others, connected to the system, are executing address instructions set in their respective track address memory. It is possible to issue a new track address instruction to successive drum files every 15 milliseconds. When the drum file completes its positioning instruction, a unit-ready signal is returned to the drum control unit, whereupon the read or write circuits of the flying-magnetic-recording-head may be activated. A select line determines whether information will be transferred to or from either the bottom or top drum.

ENGINEERING INTERESTS

A few of the components which might be of further interest are: 1) the flying magnetic recording head construction; 2) the lever adder; and 3) the servo circuits.

FLYING MAGNETIC RECORDING HEADS

Components and an assembly of the RANDEX II flying head are shown in Figure 7. The head body consists of two pieces of aluminum, fabricated to exacting dimensions. The surface of each of these pieces, which after assembly become the flying face of the head, are faced with tungsten-carbide. After the read-write coil, erase coil and common I-piece are secured in their respective cavities in the two aluminum sections, these sections are bolted together, with the joint located at the read-write coil gap-line. The gap-lines on the read-write and erase heads are spaced 0.020 inches apart. Since the pivot axis location is very critical to flying-head performance, bearing holes are accurately machined at a prescribed location on the sides of the head.

Using the bearing bores as a locating reference, the assembled head is then mounted in a lapping fixture and the tungsten-carbide face is lapped on a wheel, which gives the flying face of the head a radius of curvature somewhat larger than that of the drum. The lapped curvature on the head lies in the same plane as the drum recording surface, and must be parallel to the bearing axis.

After thorough cleaning and inspection, terminal boards are added; wiring of the coils is completed; caging cams, which retain the head when in the non-flying position, are added;

and, finally, bearings are installed.

A complete inspection to check flying and recording characteristics is then performed on the head.

LEVER ADDER

A lever adder, removed from its container, is shown in Figure 8. It is a mechanism which provides, through linkages, additional head positioning by moving the servo-rack incremental distances within adjacent teeth. Throughout this discussion it should be kept in mind that the biquinary (5, 4, 2, 1) computer code is used. In addition, a 10 has been added to the lever adder logic.

The lever adder used on RANDEX II is a mechanical adder which can position its output link to any one of 23 equally spaced positions, designated here as 0 through 22. The output motion is generated by strokes of five individual solenoids, which drive through an arrangement of levers and interconnecting links, as shown in Figure 9. All solenoid links have identical stroke lengths, however, the weighting introduced by the levers and interconnecting links is such that each solenoid produces a different displacement in the output. Specifically, in terms of output link displacement, solenoid 1 produces 1/22 the total output; solenoid 2, an output of 1/11; solenoid 3, an output of 2/11; solenoid 4, an output of 5/22; and solenoid 5, an output of 5/11. The distance that the output moves between its zero position, when none of the solenoids is energized, and its 23 position, when all of the solenoids are energized, is a summation of individual outputs.

In order to understand the manner in which the weighting produces the required motion of the output link, refer to Figure 9, and consider the case in which solenoid 1 is energized and the other four solenoids are de-energized. Solenoid 1 moves its end of the 3-lever a distance of 1 unit (equivalent to the solenoid stroke length). The other end of the 3-lever remains stationary, since solenoid 2 is de-energized. Since the 3-lever output link is 1/3 of the distance from the stationary end to the moving end, it moves

$$1/3 \times 1 = 1/3$$

unit. The 3-lever output link imparts this motion to its end of the 12-lever. The other end of the 12-lever remains stationary, since it is connected to the output link of the stationary 9-lever, which does not move because solenoids 3 and 4 remain de-energized. The output link of the 12-lever is 1/4 of the distance from the stationary end to the moving end, therefore, the output link moves $1/4 \times 1/3 = 1/12$ unit. The 12-lever output link imparts this motion to its end of the 22-lever. The other end of the 22-lever remains stationary, because it is connected to de-energized solenoid 5. Since the output link of the 22-lever is 6/11 of the distance from the stationary end to the moving end, it moves

$$6/11 \times 1/12 = 1/22$$

unit. By a similar process of reasoning, it can be verified that each of the other solenoids introduces a motion corresponding to the 2, 4, 5 and 10 digit with which it is associated, divided by 22.

In the RANDEX application only 20 of the 23 possible positions of the lever adder are used, therefore, the total distance that the output link is moved is only 19/22 of the solenoid stroke length. Space must be left between position 19 of one tooth, and position zero of the next adjacent tooth to obtain equally spaced information tracks at a density of 20 tracks per servo-rack tooth. Total rack motion is then limited to 19/20 of the distance between adjacent servo-rack teeth. A solenoid stroke length, equal to the distance between adjacent teeth, was selected for this design. Therefore, a differential pulley, connected between the lever adder output link and the servo-rack, is used to multiply the output motion of the lever adder by a factor of 11/10. In the example above, the servo rack moves

$$1/22 \times 11/10 = 1/20$$

of the distance between adjacent teeth.

By varying the differential pulley ratio and lever weights, other computer codes can be used with the same basic lever adder assembly.

Primary features in the construction of the lever adder are the levers and dashpots, which are located between the solenoids and output.

The levers are manufactured as segmented circular discs (See Figure 10). Note that the link (thin band with pin on end) wraps around a circular portion of the lever to its point of attachment to the lever. This is done to provide constant spacing between links, simplify manufacturing, and reduce link stresses. Relative positioning of the circular portions determines the lever ratio.

As solenoids are energized and de-energized, the associated dashpots control the acceleration and deceleration of the rack and lever adder system. To function properly, the entire lever adder is put into, and kept in, a container filled with damping fluid. A series of holes toward each end of the cylindrical section of dashpots allows the passage of damping fluid into and out of the cylinder.

Since upward and downward movements of the dashpots piston are symetrical, only the downward movement will be discussed. The lever adder output is biased by spring tension on the servo-rack. When a solenoid is actuated, the

plunger pulls in rapidly, stretching an interconnecting spring between the dashpot piston and plunger, as a result of the piston being restricted in its movement by the damping fluid in the dashpot. The pull, exerted on the piston by the spring, moves the piston which results in damping fluid being forced out through the hole in the wall of the dashpot. As the piston moves beyond the midpoint of the stroke, it starts to cut-off the openings, thereby reducing the escape area for the damping fluid remaining in the cylinder. Size and location of these holes control the piston acceleration and deceleration. Since the servo-rack is connected to the dashpots through links and levers, its motion is also controlled. When the piston reaches the end of the stroke, very little kinetic energy remains in the system, and overshoot or jerk is virtually eliminated. Bias springs, including the servo-rack spring, move the dashpot pistons in the opposite direction as solenoids are deenergized.

SERVO

The servo cycle consists of a "forward torque" phase, a "closed loop" phase, and a "reverse torque" phase.

During the forward torque phase, the servo control logic connects the forward torque output of the torque circuits transformer to the servo amplifier. The phase of this fixed amplitude signal is such that the carriage is driven to the right, away from the tooth with which the carriage pawl is initially engaged. At the same time, the holding signal is removed from the pawl coil, and the pawl retracts by spring tension, thereby clearing the teeth of the servo-rack. The duration of the forward phase is timed by a delay flop in the servo logic.

At the end of the forward torque phase, the servo control logic begins the closed loop phase by connecting the error signal from the servo potentiometer arm to the servo amplifier. The amplitude of this error signal is proportional to the distance between the carriage position and the new address position. The phase is dependent upon the direction of the error. The carriage is driven to the position midway between the addressed tooth and the next tooth to the right, at which position the error signal is zero. In order to prevent the carriage from excessively overshooting the required null position, a signal proportional to the velocity of the carriage is combined with the error signal. The error and velocity signals are supplied to a "zero detector" circuit in the control logic, as well as to the servo amplifier input circuit. When the error and velocity signals reach zero, the control logic terminates the closed loop connections and sets up the reverse torque connections.

In the reverse torque phase, the reverse torque output of the torque circuits transformer is connected to the servo amplifier. This is a fixed amplitude signal, phased so that the carriage is driven toward the addressed tooth. At the same time, pawl control signals from the control logic circuit cause the carriage pawl to be extended. Thus, the pawl is driven against the addressed tooth. The application of the reverse torque to the servo amplifier input circuit is timed to coincide with the peak amplitude of the reverse torque voltage, thereby lowering the response time of the servo. In addition, in order to allow a period of undamped acceleration, the application of the velocity signal for damping to the servo amplifier input circuit is delayed for one cycle of the reverse torque voltage. The velocity signal is then applied, resulting in a reduction of the velocity at which the carriage pawl is driven against the addressed tooth. The reverse torque phase continues until a new servo cycle is initiated.

REFERENCE

[1] J. P. Eckert, J. C. Chu, A. B. Tonik, W. F. Schmitt, "Design of UNIVAC® -LARC System: 1," EJCC, Proceedings, 1959. p 61.

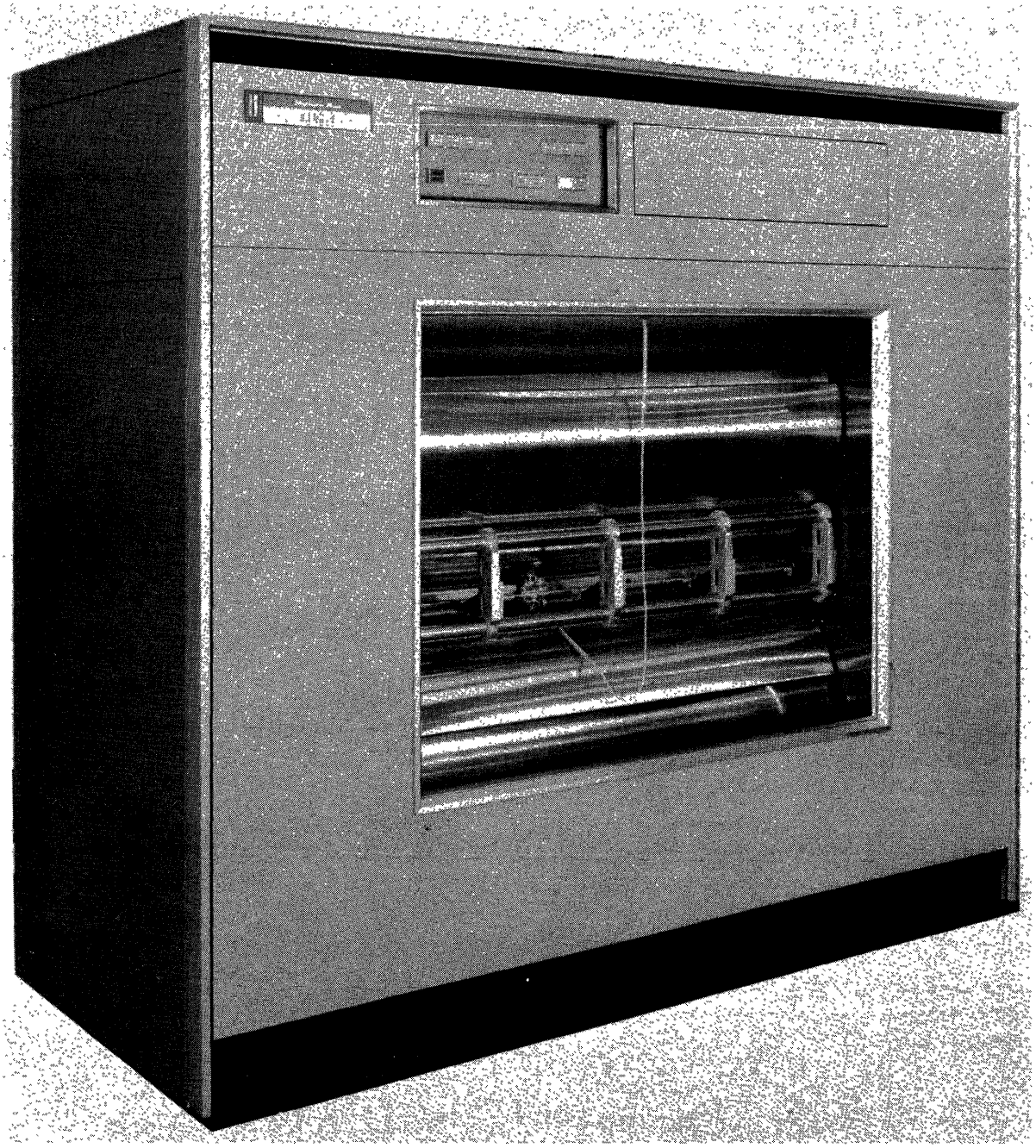


Fig. 1. RANDEX Drum File

STORAGE

TOTAL STORAGE	198.588 million bits
INFORMATION-UNIVAC	126.72 million bits
SOLID STATE COMPUTER	25.344 million digits

DRUM PARAMETERS

LENGTH	44.0 in.
DIAMETER	24.3125 in.
SPEED	870 RPM
BIT DENSITY-USSC	650 PPI
BIT FREQUENCY-USSC	720 KC
TRACKS PER DRUM	2000
TRACKS PER INCH	50

ACCESS-FLYING MAGNETIC HEAD

MINIMUM HEAD POSITIONING TIME	125 millisec.
MEAN HEAD POSITIONING TIME (ASSUMING RANDOM ADDRESSES)	350 millisec.
MAXIMUM HEAD POSITIONING TIME	550 millisec.
MEAN DRUM LATENCY TIME *	35 millisec.
MAXIMUM DRUM LATENCY TIME *	69 millisec.

RECORDING

READ-WRITE ELEMENT WIDTH	0.011 in.
ERASE ELEMENT WIDTH	0.019 in.
HEAD TO DRUM CLEARANCE	0.0002 in.
HEAD POSITIONING TOLERANCE (PLUS-MINUS)	0.002 in. max.

DRUM FILE PHYSICAL CHARACTERISTICS

LENGTH	76 in.
WIDTH	33 in.
HEIGHT	68.5 in
WEIGHT	2000 lbs.

ELECTRIC SERVICE

AC INPUT TO DRUM FILE-60 CYCLE	
KVA	
STARTING SURGE	5.5
RUNNING	2.0
VOLTS	230
DC INPUT TO DRUM FILE SUPPLIED BY DRUM CONTROL UNIT	

POWER DISSIPATION

AC	1600 watts
DC	500 watts

COOLING

HEAT DISSIPATED	9200 BTU/hr.
MAXIMUM ROOM TEMPERATURE	100°F
MAXIMUM ROOM HUMIDITY	90 percent

* Time for data of a specific track to appear under the head once the head has reached address.

Fig. 2. RANDEX II Specifications

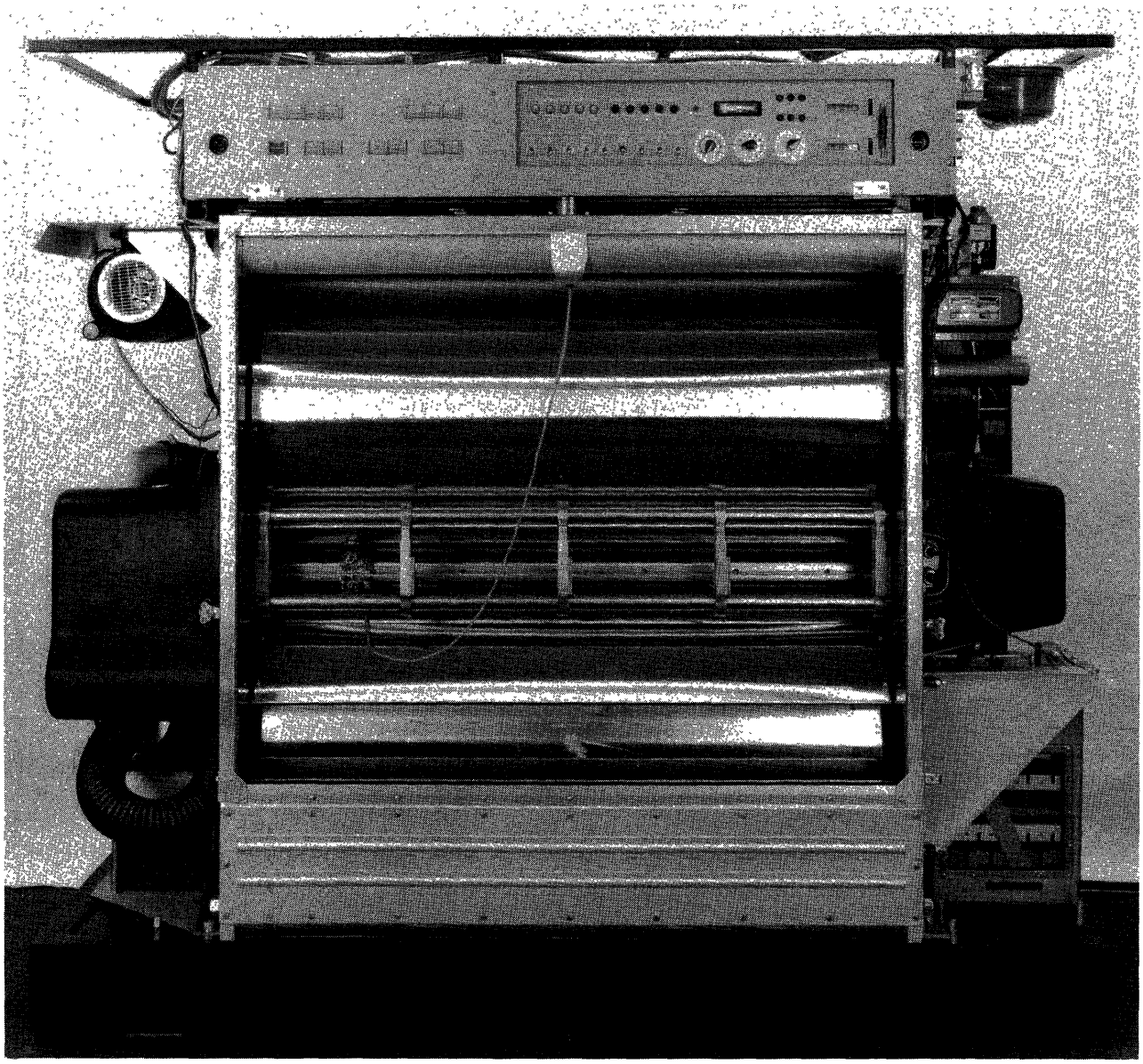


Fig. 3. Front View, Less Casework, Showing Sealed Enclosure

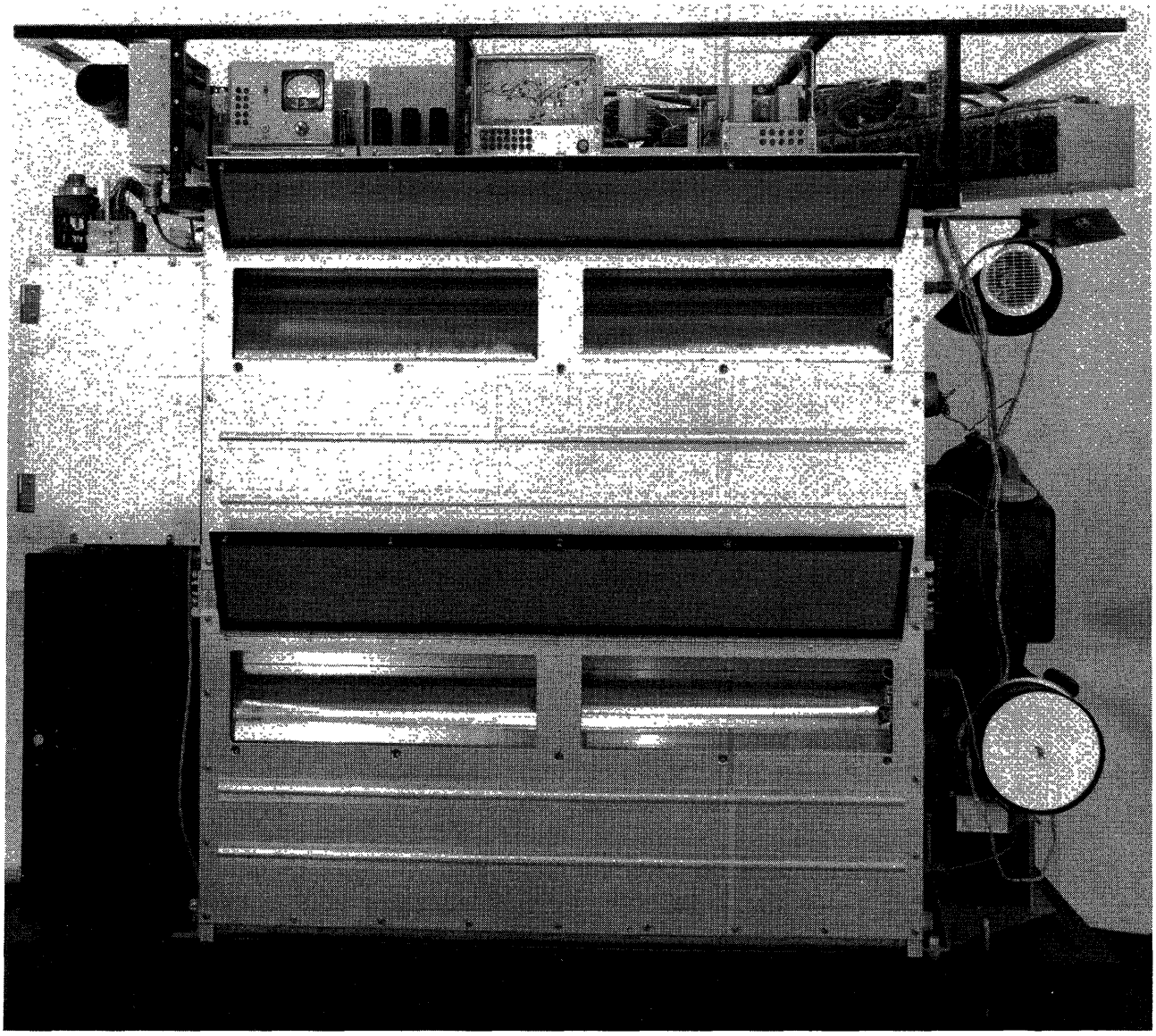


Fig. 4. Rear View, Less Casework, Showing Access Areas

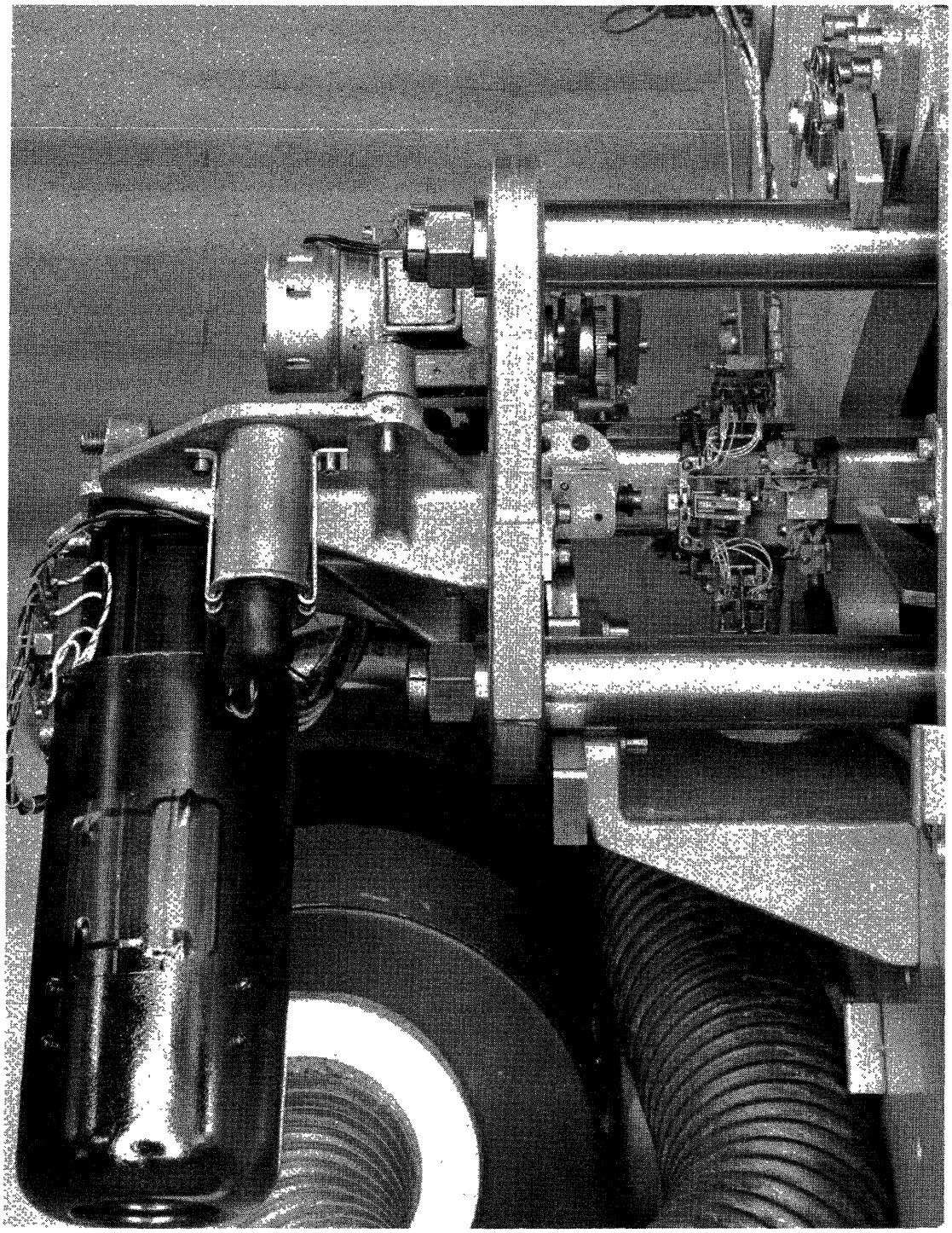


Fig. 5. Servo and Carriage Access Area

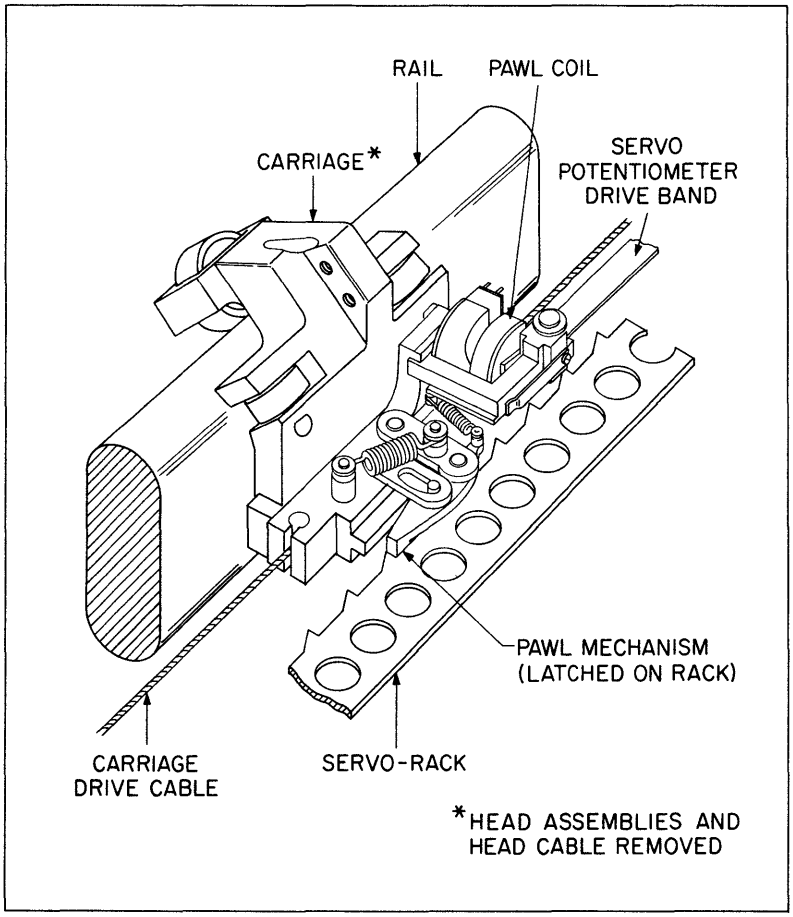


Fig. 6. Carriage, Rail and Servo – Rack Arrangement

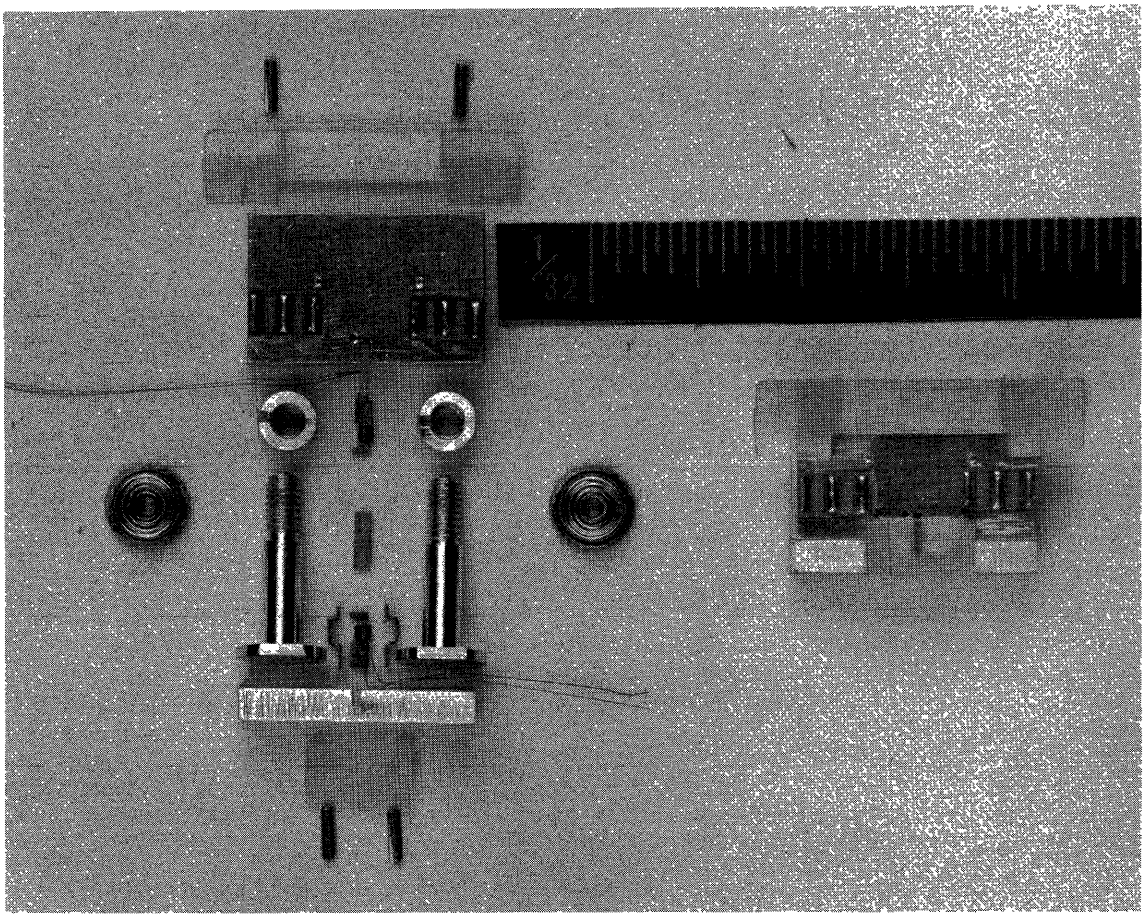


Fig. 7. Flying-Head Components and Assembly

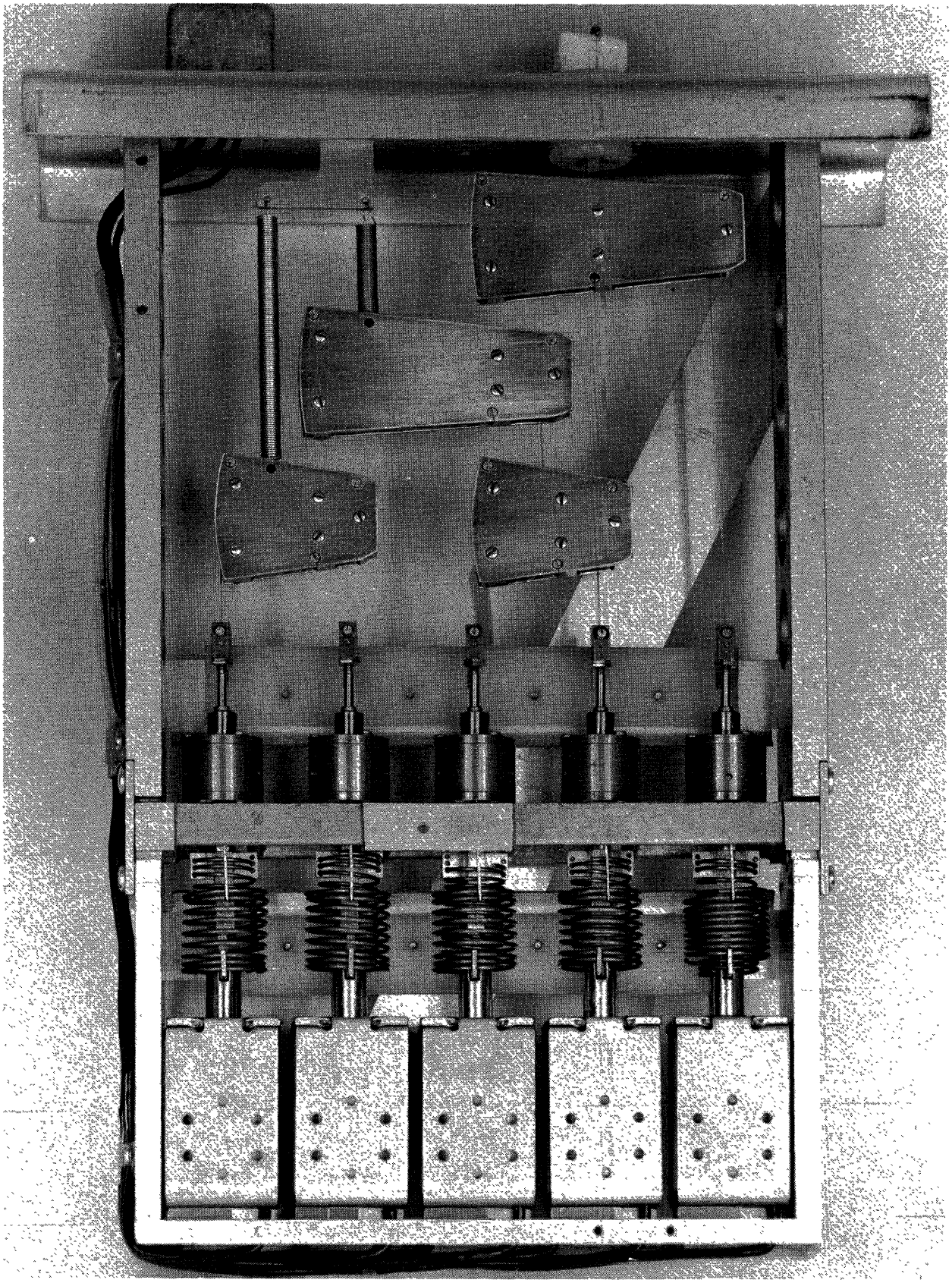


Fig. 8. Lever Adder Assembly

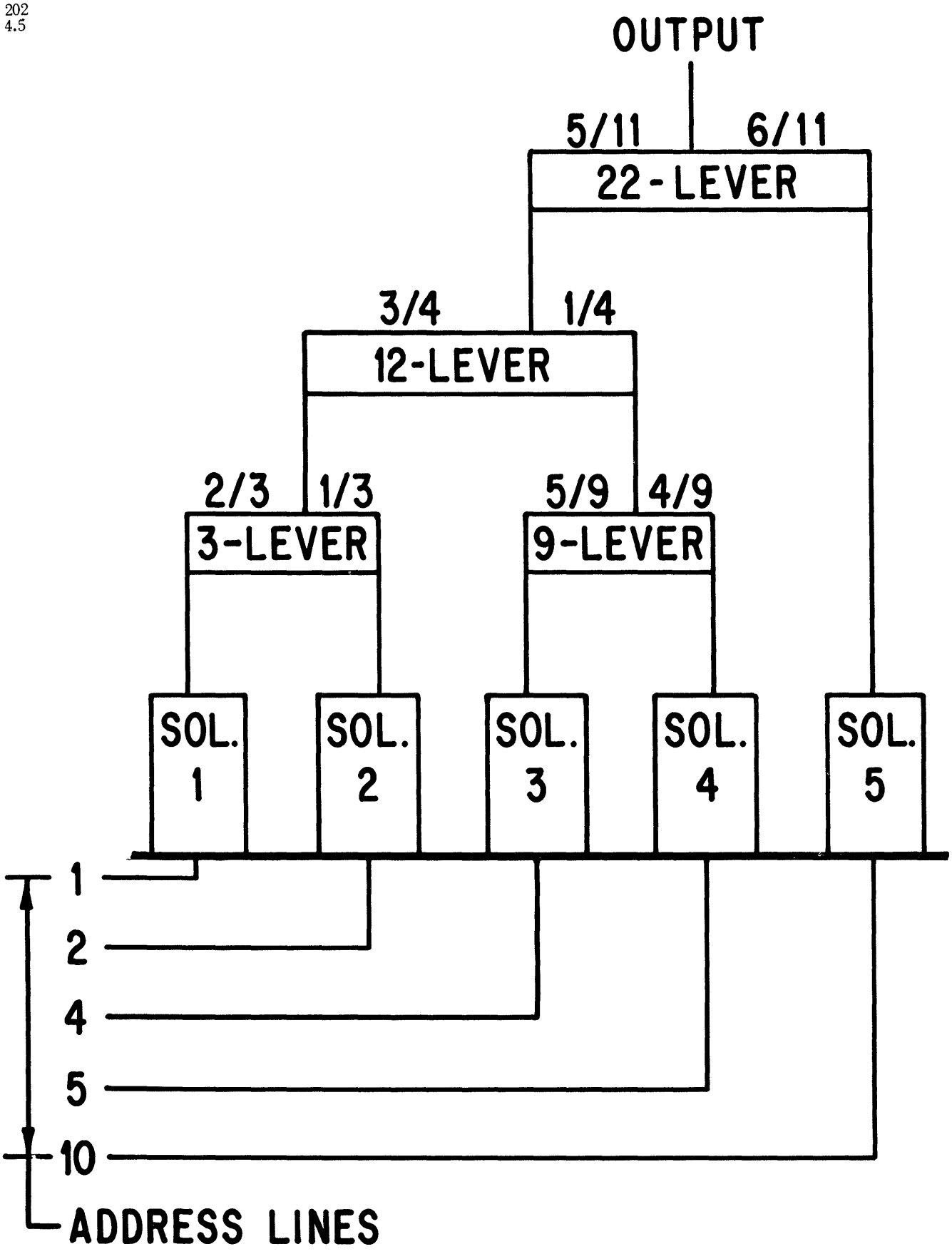


Fig. 9. Lever Adder Schematic

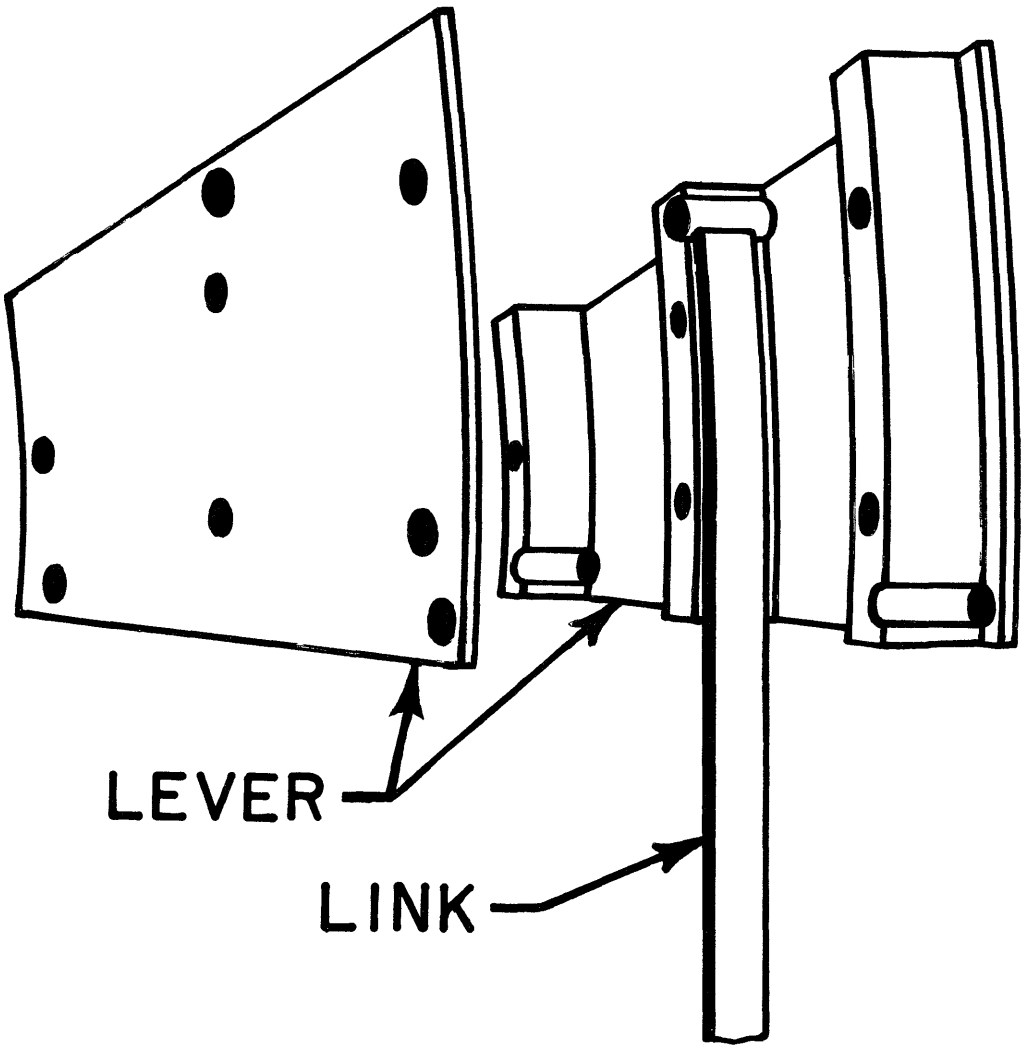


Fig. 10. Lever Assembly

DATA PROCESSING TECHNIQUES IN DESIGN AUTOMATION

By Dr. William L. Gordon
Minneapolis-Honeywell Regulator Company
Electronic Data Processing Division
Newton, Massachusetts

Summary

By providing a computer with basic information concerning the design of a device as complex as the modern computer one not only obtains an efficient record retention system but also brings to bear the full decision making abilities of the computer on the design problem itself. A major thesis of this paper is that the automation of the design of a complex system is primarily a data-processing problem in which the most powerful tools reside in the ability of the computer to perform such jobs as editing, extracting, sorting, and merging pieces of basic design information. This contention is substantiated by briefly describing the system currently in use to provide mechanized aids to design and production at Minneapolis-Honeywell Regulator Company, Electronic Data Processing Division.

Introduction

In the course of proceeding from the design to the construction and maintenance of a device as complex as the modern computer one is led rapidly to the need for efficient record retention systems capable of handling such diverse pieces of information as the formal logic of the device or the production wiring specifications. Not only must the record system be capable of rearranging and presenting the information for particular use, but it desirably must also do what it can to reduce the gigantic clerical job of effecting the transition from the basic design considerations of logic and circuitry to their amalgamation in the finished machine.

The major advantages of such an automated design system lie in its ability to perform rapidly and accurately while busily exploring the consequences of basic design changes, however trivial. The fact that such automatic processes do exist is additional testimony to the systematic approach to machine design so essential to the human consideration of any complex device.

This paper is a report on such a Design Automation System currently in successful use at the Honeywell Electronic Data Processing Division. It is concerned primarily with the techniques of utilizing the capabilities of an electronic data processor, in this case the D-1000. The inputs to this processing system consist of statements of the purely logical design of a unit of arbitrary size, skeletal information as to logic and circuit placement, and descriptions of the individual circuit types. Outputs are quite varied and range from analysis of the logical design to wiring specifications for the machine; this latter including punched card decks to operate automatic wire wrapping machinery.

The program system was originally executed to aid in implementing the design and construction of the H-800 and will find use as well in the production of future systems. This latter statement should be not interpreted as saying that future designs will necessarily be forced into the same mold, but rather as asserting that the processing system is of sufficient flexibility and simplicity to guarantee rapid adaptability to changes in basic design philosophy. Indeed, with a few data safeguards removed, we would be quite capable of specifying a length of coaxial cable between two remotely located insurance policies.

The most distinguishing feature of the program system in operation is its low usage of the arithmetical abilities of the computer. This is conditioned less by the fact that the D-1000 is most powerful when employed with the business data-processing operations of sorting, merging, and high speed information flow to and from magnetic tape as by the fact that the job at hand was a data processing job with the model found. It is a major thesis of this paper that Computer Design Automation (as well as design automation of other complex systems) is principally a job of symbol manipulation lying almost entirely within the realm of standard data processing technique. The reader will undoubtedly agree with this statement insofar as record retention is an aspect of the design problem. The point to be made here is that the design process itself is largely a process of augmenting and expanding basic information by a series of merge (match) passes, extraction, editing, or data rearrangement passes (sorting), over the basic information.

Basic Processing

Input

Logical design data is presented to the machine in the form of Boolean equations modified to meet punched card requirements. Each logical statement carries with it an indicator of the circuit type to be employed in the physical realization of the statement. This latter device enables the designer to specify the input-output relations of non-logical or time-sensitive devices for which Boolean notation is inadequate, the circuit type designator serving to remind

the reader (and the processing program) of the way this black box should be treated. For example, the statement:

(1) GBA A/B.C * D.E.F

makes logical sense if one interprets "/" for "equals", "." for "and", and "*" for "or", but may more simply be interpreted as establishing, at the very least, an input-output relation between signals B and A, C and A, etc. Indeed, a possible way to file away the logic is in this latter form. However, given a file of logical statements in the form (1) it may be easily transformed into the latter by a single pass over the data. This leads us immediately to the process of examination of the loads on all signals....a process typical of machine use in this system. (cf. Figure 1)

Signal Loading

The original logical statement is regarded as specifying a source together with its inputs. Dually, each input may be regarded as a load on the source bearing its name. By forming an item for each occurrence of each literal the logic may be recast into a file of signal name pairs indicating a relation of the form "B is an input to A". Sorting all such items on the input field groups all occurrences of a signal name together thereby forming a list of all loads on all signals. The driving signal name itself is properly treated as an input to an unknown load. With auxiliary information descriptive of the circuit type and gating structure dragged along for a free ride, a final pass editing for printout has an easy time tallying loads on each signal.

Matching the tape forming the input to the sort with the sort output tape forms a simple means of cascading each signal through another level of logic. This is an effective method of chaining the input relation through as many stages as desired. By reinterpreting the relation as meaning "is a part of" this procedure is seen to have a direct counterpart in the business of exploding parts inventories on the basis of assembly levels.

So far, this has examined the logic from a purely abstract point of view (quite proper to logic) but the designer is not merely playing games on paper. These logical statements are intended to have a circuit realization. Again a dual statement may be made, i.e., the circuits are to have a logical realization. With this in mind a master file of available circuit (package) types is treated in like black box fashion with a typical entry appearing as:

(2) GBA 10/11.12 * 13.15.17

where physical pin numbers or other designators indicating location relative to the package at hand replace the formal logical symbols. This device of using the same format forces the logical aspects of the circuit description into the same mold as the logic itself. Aside from appearing a natural way of representing the circuit it leads easily to the production oriented step of determination of wiring configurations.

Network Determination

The formation of a file in which all pins of the same wire network are grouped together is identical to the process used in loading calculations with the exception that a correspondence between the symbolic logic and the package structures must be made first. This is done via the use of a third file specifying for each signal the type of package it is being implemented with and the slot location of the package in the racks. The logical statements are matched with this last file, the resulting tape resorted by package type designator and then matched against the pin structure. Resorting the final output produces a file organized by logical signal name in which the symbolic literals have had package and pin locations associated with them in one-to-one fashion. The resulting document is a compendium of the entire design and all other information is derived from this generated file. Reprocessing the pin modified logic through the loadlist type extraction and sort regroups all occurrences of each signal. The pin locations that here ride along are likewise grouped together thereby forming a first approximation to a wiring list.

It should be remarked that the processing system to this point is sensitive only to the names of signals and location designators and has not found it necessary to establish any geometric relations between the physical points. In this sense no mathematical computation other than tallying has taken place. Nevertheless, the original logical design has been exploded and recast into a form which could be used for construction purposes. Indeed, a highly simplified version of this system was implemented with tabulating equipment and employed in the successful construction of a small machine.

Wiring Determination

The principal remaining problem is to pair the pins of a given signal run to form wires. Here simple sorting is no longer effective, for the circuit designers have determined that a desirable criterion for wiring a given network is to minimize the total wire length in the entire run. It is at this point that geometry rears its head and metric relations between pins must be examined. Nature and Mathematics still smile however and this becomes one of those rare cases in analysis where doing the best thing locally, i.e., building up the network iteratively by appending closest points at every step, produces a demonstrably minimum overall solution. With further constraints (e.g., no more than three wires may be commoned at a particular pin) the algorithm breaks down; but again Nature is benign and the deviations turn out to be statistically insignificant.

It is this process, computational in character, that forms the largest fly in the ointment of the thesis that automated design consists almost exclusively of information shuffling. Consequently it will not be reported upon in detail here, but is certainly of sufficient interest to warrant independent presentation. Overall system flexibility is still retained by isolating the length decision-making algorithm with a routine which interprets all pin designators in terms of an absolute 3 - dimensional coordinate system thereby allowing complete revamping of the construction geometry with small programming changes. A portion of this wire determination system includes further computation for the machine selection of the various wire types to be used; which selection procedure accounts for both wire length and circuit loading.

With wires fully determined, additional information as to capacitative loading, current flow in each wire branch, number of wires per pin, etc., may be easily accumulated and reported to the designer who may make necessary modifications, update his files and come through the entire system again. When he is satisfied with the design the wiring file for the unit is ready for release to production.

Production Outputs

At this point the file of wires may be broken down and sorted six ways from Sunday to fit production jigging and assembling requirements, (left-handed assemblers need not apply, for the information is presented for assembly by right handed operators). With all breakdowns and sorts made from the parent wiring file, accuracy is guaranteed up to the point of wire insertion. Independent lists are provided for inspection and ring out purposes.

The use of automatic wire wrapping machinery in the production process has necessitated the construction of a program system to effect the translation of the wiring information to a deck of punched cards which will instruct the off-line automatic equipment. The value of the data rearrangement is high in this case because the problem of sequencing the wire wrap machine for efficient operation requires great reordering of the wires in terms of such criteria as path configuration, color, length, number of wraps on pin, etc. Again this particular work will be reported upon elsewhere.

Manual Entries

The only input point indicated to the system has been at the level of the basic logical design. Although minor changes can be exploded through the entire system it has been found desirable to allow the engineer access to the output wiring files. Manual intervention at this point can account for the special cases not worthy of programming into the system as well as allow the engineer final say on what the results of processing have been. It further enables the engineer to make minimal changes to a machine already under construction. The nature of the decision making portions of the system, notably the point at which the wiring network is determined, make it difficult to guarantee that second passes through the full system on moderately input data will keep change minimal. This is largely because the routines are trying to minimize an overall criteria.

Checking

With manual modification possible the processing system continues, but in a different mode. Now it cannot generate final information, it can only check on the apparent accuracy of the file. Two modes are still left open to perform these checks. The first is that after updating the wiring file a check can be made on connectivity of each electrical path. The second check to be made on the modified wires is to do a complete reversal on the original processing system and regenerate from the wires the logic that they implement. This latter file is the true document of the machine incorporating all known information about the design. Unfortunately it is still incomplete for it merely describes what the machine should be - as opposed to what it really is; this latter anomaly occurring only because the insertion of the wire into the machine and into the machine file are two different processes.

Extensions & Generalizations

The basic model exploited in the above system is quite simple, the computer is regarded as a massive assemblage of "black boxes" (circuits or packages) each possessing input and/or output points connected together by a relational scheme known only to the logical designer. The individual black boxes themselves are constructed as an assemblage of other "black boxes" (components) according to a scheme known only to the circuit designer. We may continue up or down in this hierarchy to include the system designer or the component designer at macro-molecular or subatomic levels, but for convenience and with no real loss of generality will settle at the level of the circuits which the logical designer must interrelate. Each such circuit has well defined input and/or output points which in the finished machine accumulate a large number of attributes, among which are the following:

1. A designator indicating the relative position of the point in the machine (e.g., connector pin location).
2. A designator indicating the type of circuit to which this node belongs (e.g., flip-flop).
3. Number of connections made to this point.

4. Description of the wire coming to this point* (length, type, route, etc.).
5. Items 1 and 2 for the other end of the wire coming to this point.
6. Items 1 and 2 for the signal source driving the network of which this point is a member.
7. Current flow in the wire of item 4.
8. Wave forms at this point.
9. Name of the signal appearing at this point - name of the network.
10. Structural significance of this point in relation to the inside of the black box. (e. g., 5th leg of the 2nd gate etc)
11. Name of the black box this wire is going into.

The above list is by no means exhaustive, nor does any person associated with the design, construction or use of the machine have a need to examine all of the information contained therein at any one time. By the same token it is not until the design is complete that such a compilation may be attempted for each node (pin) in the device. The major function of the Honeywell Design Automation System is to piece together the essential pieces from various design sources and at some points compute a few of the items in the list.

In examination of the list more closely several items may be grouped as falling into distinct design areas. Items numbered (2),(9), (10), and (11) suitably edited form the logical design; whereas item (10) above represents the link between logic and circuitry. Combining items (9) and (1) is essentially the job of package allocation. Matching these pieces of design information produces input to a wiring determination routine from which all else is derived. Curiously, in the course of back-tracking from wires to logic the program system at one point uses logical structure as a key for sorting.

The present system is keyed heavily to the problem of backboard wiring interconnection of the circuit "black boxes". An enlargement of the system will enable machine processing to go inside the packages or outside to the major unit complex. Here we will be exploiting the fact that all location designators are relative to the level at which they occur. For example, each component placed on a package is located only on that package; each package inserted into a slot in the machine is located only relative to that unit (e.g., peripheral control unit).

* Wiring networks are usually simply connected, i.e., there is only one wire path (equipotential) between two points. A convention establishing one point in such a network as a source establishes an orientation for the two ends of each wire in the network.

Remarks

The generality of the above model suggests its application to other equally complex jobs. All we have really done is endowed each connection point with a name and address obtained from different sources. Relations between names are interpreted as calling like relations between addresses; indeed the very application of Boolean Algebra to switching circuit theory came about by making symbolic name assignments to physical circuits, largely divorcing the problem of logical design from its physical implementations. Certainly the same technique is applicable to other complex systems ... the procedures described here effecting the reverse step of making all necessary name and address correspondences. This degree of abstraction has played a significant role in computer design and use, enabling the breakdown of a complex system into understandable smaller pieces; or rather the assemblage of the system from comprehensible building blocks.

Name and address processing is certainly familiar to the programmer who in an effort to free himself from physical address considerations has constructed automatic program systems to allow writing his program symbolically (as the logical designer does today) letting the machine solve his allocation problem. The only essential difference here is that when this is done he rarely has to communicate in detail the resulting information to anything other than a computer.

Acknowledgements

The system outlined here is not something that springs full blown in either conception or execution. The author is indebted to Dr. J.J. Eachus for proposing this as a data handling problem, to Dr. E.J. Dieterich for his advice, criticism and direct programming aid, and to the members of the Systems Analysis Division of Engineering for yeoman work in devising a working program system. Both Engineering and Production divisions of Honeywell EDP must be thanked for their ready cooperation in the effective use of the system.

The problems discussed here involve processing technique only. Such a working system must also cope with the off line problems of file and document control. The major difficulties that arise are in areas of human communication, for the design automation system has executed a heavy influence on engineering and production procedures. Analysis and programming are relatively mild aspects of the problem; but perhaps this is because machines are really quite docile.

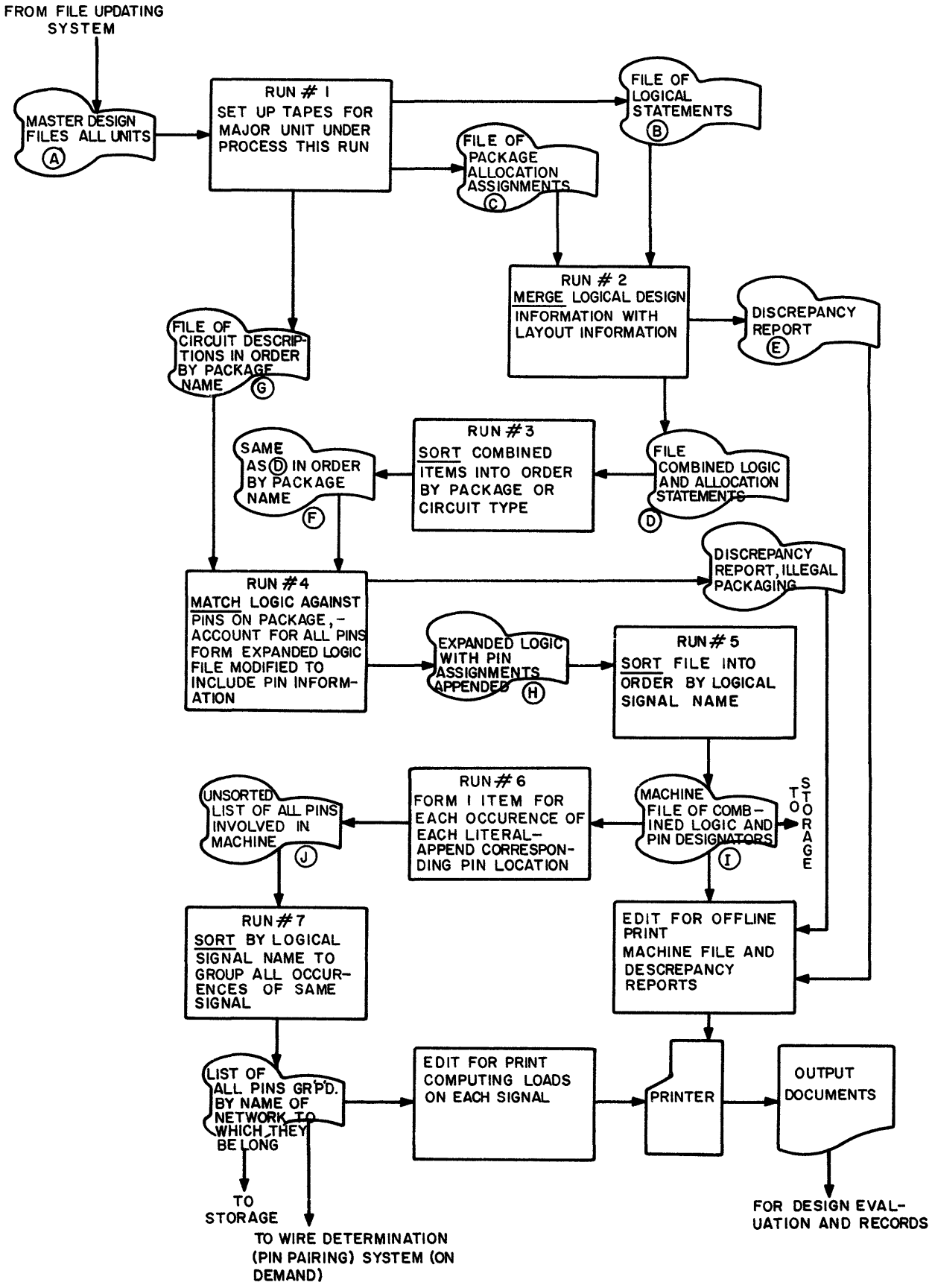


Fig. 1. Process Flow for Generating Pin Groupings from Formal Logic and Packaging Information.

IMPACT OF AUTOMATION ON DIGITAL COMPUTER DESIGN

by
W. A. Hannig and T. L. Mayes
General Electric Company
Computer Department
Phoenix, Arizona

Introduction

A digital computer logician's job may be defined as a two-fold task. First, he must generate abstract ideas on the organization of a proposed computer to handle a particular type of problem, and second he must convert these abstract ideas into data that a factory can use to construct and check out the computer.

At the General Electric Computer Department, we have spent considerable effort in supplying tools for the logician to use to simplify and to better optimize the job of converting abstract organizations into factory data. These tools are automation programs which allow the logician to use programs, run on existing computers, to design proposed computers.

The paper describes:

1. The functions performed by these programs;
2. The logician's use of these programs as a tool;
3. The use of the data produced by these programs; and
4. The effect that the use of these programs has upon the human organization that designs and builds these computers.

The Functions Performed by the Programs

Summary of functions

The programs may be divided according to five major job categories:

1. Arrangement of Boolean equations to describe the over-all computer logic.
2. Interpretation of Boolean equations to determine logic gates and other circuits required; how they are to be interconnected.
3. Assignment of circuits to plug-in circuit cards.
4. Generation of information for interconnecting circuit cards.
5. Preparation of logic schematic diagrams.

The program functions in each category are discussed in detail in the following sections.

Arrangement of Boolean Equations (See Fig. 1a.)

For describing the over-all logic of a digital computer, it is of great advantage to have a set of Boolean logic equations which show how each flip-flop is controlled during each time interval of each command. Such equations are assembled and arranged by this program from a manually prepared list of the control signals active during each interval.

In the execution of a command, various control signals, emanating from a command decode network or from controlling counters, enable various gates in a series of steps to execute the command. Thus, for each command the logician lists the control signals which are turned on during each step of the execution. This list, in punched card form, is sorted and printed in order of command step and also in order of control signal (to show the command steps using each control signal). The list in its former sequence is one of the inputs to the program.

The logician also prepares on punched cards a list of the logic equations which are activated by the control signals referred to above.

And, finally, the logician prepares special program control information which deals with the identifying headings. These headings will appear in the printed output, which deals with numerous exceptions to the pattern (such as the implication of equations due to the absence of a control signal during a particular command step), and which organizes the over-all program logic.

From these three inputs the program collects the equations for each command step. The equations for each step are collected by searching the equation list (on magnetic tape) for the equations corresponding to the control signals associated with that step. With suitable headings interspersed to identify the various registers, the collected equations are subsequently formatted for off-line printing on regular drawing forms. By careful control, all the active equations

for one command step generally fit onto one sheet to make the results easier to use. (Example in Fig. 2.)

Each major, independent equipment of the system requires this logical description information by preparation of lists of control signals, equations, and program control information. The three kinds of data, prepared by the logicians, are processed by one set of routines, virtually unchanged from one equipment to the next. This characteristic is generally true of the other programs as well; the variations represented by the different major equipments were treated simply as variations in the data processed by the programs.

Interpretation of Boolean Equations (See Fig. 1b.)

The original equations discussed above are grouped together manually to form the complete "unimplemented" equations for each flip-flop. The logician then operates upon these equations to match the available logic circuit structures and circuit rules. He expresses his results in the form of "implemented" logic equations which are punched into cards. (Example in Fig. 3.)

These equations are interpreted by a program in order to produce a list of the "elements" required to implement the desired logic. An "element" is the smallest subdivision of a computer which still retains its logical identity, e.g., AND gate, OR gate, flip-flop, etc. Each element is described by a unique code name and by a list of the input signals to the element. Thus, this program produces an Element Input List which identifies all the elements required and shows how they are interconnected. These elements are free-floating; they are not associated with plug-in cards as yet; hence, no pin numbers are present.

The process of interpreting the equations is related to other equation interpretation programs in which the equation is scanned and thereby divided into unique entities, in this case, the elements. Each equation contains signals and symbols (logical operations, parentheses, and such). Beginning from one end of the equation, the program selects the first three of these, regardless of whether they are signal or symbol, and classifies all signals as simply another unique kind of symbol. The three-symbol group is looked-up in a table containing all possible combinations of three symbols. If the group is permissible (two plus signs together would be impermissible), a "level increment" is obtained and algebraically added to a cumulative level number. The table also supplies information as to whether the resultant level number should be odd or even and what to do to it if the number requires adjustment. When finally acceptable, the level number is stored in association with the middle symbol of the group (the first and last symbols of the equation are asterisks).

The same process is repeated for the next group of three symbols, using the second and third of the previous group as the first and second of the next. Thus, a two-symbol overlap occurs between successive groups. In each case the cumulative level is adjusted and assigned to the middle symbol until the entire equation has been traversed. In some cases, a prepass is necessary to ascertain the presence of certain kinds of symbols.

Following the level assignment, the program, still working on the same equation, successively scans the level numbers to detect particular values grouped together. The signals within such a group are the inputs to an element, and that element is immediately identified and added, with its inputs, to the list of elements being prepared. The list of symbols, signals, and level numbers is also modified appropriately. In this process, successive levels of gates are identified, starting from the outside gates of the structure and continuing until the innermost gates have been identified and set up.

The program complains if the number of levels implied by the equation is excessive for the particular type of "source" element (which the gates, as a group, control). The program shifts the gating structure about somewhat in accordance with certain limitations imposed by the circuitry. It also inserts degenerate (one input) gates as required by the circuits rules. Gate width is checked as a function of the location of the gate in the structure and the type of source element involved.

The resultant list is, in a sense, a list of equations having only one level each, where the logical relationship is identified by the kind of gate. But rather than view them as equations, they are viewed as elementary building blocks, the distinct entities of which comprise the computer being designed. This list is the main stream of the programs which follow. The Element Input List is

operated upon in various ways; other lists are derived from it, but it continues to appear along the line in altered, augmented form until it represents a complete determination of the machine from which manufacturing information and reference documents are derived.

One of the uses of the Element Input List is to serve as input to the loading calculation routines which prepare a sort item for each element's input signal whose loading must be evaluated. These items are sorted to gather together the names of the load elements on the sources being evaluated. Through the use of a manually-prepared table showing the location of the registers (in a gross sense), the program computes the wiring capacitance. Using a table of a-c and d-c loading limits and information contained in the load list on timing restrictions (derived from the input logic equations) in which critical and non-critical timing is identified, the program assigns the loads to the sources in such a fashion as to group the loads geographically and to avoid overload. Parallel sources (producing identical logic functions from identical input signals) and the opportunity for the program to add power drivers where the timing permits, impose a relatively complex assignment task. Each time a load is added, for example, the wiring pattern may change greatly or very little; a great change may jump the capacitance appreciably and prevent the addition of the load. Thus, the wiring must be routed and its length computed repeatedly.

Once the loads have been assigned, the program determines the pull-up or pull-down currents required to drive the loads properly and produces a list of data showing which specific source drives each load as well as a list of the drivers added. These lists are used to update the Element Input List, which previously has shown source signals in a generic sense only, rather than by specific element name. The load list is also formatted and printed for use by the logician. Overload cases which the program could not cure because of timing restrictions also are listed for the logician.

For some equipments, an additional loading calculation is made to reduce the amount of driving requirement, and hence the circuitry, required. In this calculation, the loads on a source are examined for the presence of certain types of signals which would absorb all of the gate's load, and which would be mutually exclusive of each other. Introduction of such signals into a pair of loads driven by the same source would reduce the effective load on the source to that of only one load. The massive amount of timing information and considerable processing required made it imperative to do this calculation by computer. Substantial reduction of circuitry resulted, and in some cases otherwise impossible logical configurations were made possible by this approach. The assignment of these mutually exclusive loads required special treatment in that the group could not be divided among the driving sources without recomputing the effective load.

Assignment of Circuits to Plug-in Cards (See Fig. 1c.)

The next program assigns the elements to plug-in cards. The available cards are represented in skeleton form in a master file. Each skeleton includes the circuits contained on the card in the form of outputs and inputs grouped together with the associated pin numbers shown. Thus, the circuits are in essentially the same form as the elements.

Basically, the program must find a circuit of suitable type for each element and assign it by entering the name of the element as the output of the circuit and by entering the input signals of the element as inputs into the circuit. Space is provided in the skeletons for the program to enter these input and output signal names (the output signal is identical to the name of the element).

If a suitable circuit can not be found on an existing card, the program must select an appropriate card type from the skeleton file, copy it into the list of cards being used, and assign the element as described.

With numerous ways in which some elements may be assigned, including dividing them among two or more cards, the program makes use of tables so organized to provide intricate logical ability in combination with control data contained within the skeletons themselves. Some of the plug-in cards contain partial gating structures of various types, and the program must do considerable work to locate sets of gates which can make use of these structures. The programs also must look ahead, in effect, to determine the card type which should be set up not only to satisfy the needs of the immediate element being assigned - which might be satisfied in any of several ways - but also to provide

a card type suitable for the remaining elements to be assigned to this group of cards.

The group of cards dealt with together is limited to twenty-eight, the number contained within a physical module. Because a specific group of elements is not always assigned to a specific module, the program must assign and reassign increasing numbers of elements to cards until the number of cards reaches the allowable limit. The reassignment is necessitated by the changing pattern of card types and numbers as additional batches of elements are added. Sometimes, an altogether different card type may be used when elements are added to the group because of their ability to justify such a type. In another mode, a definite batch of elements is designated for a module, and the assignment becomes simpler and more rapid. In each case, a manually prepared table is used to designate the twenty-eight card modules which are to contain various groups of elements.

The resulting list of plug-in cards shows all the cards assigned by the program. Spare circuits are identified by blank spaces, because of the lack of signal names associated with the circuit identification.

This list, called the Element Pin List (EPL), is a key and permanent file. Its format permits easy modification by manual updating which is accomplished simply by preparing one punched card for each pin the signal of which is to be changed. Entire plug-in cards may be added or deleted by single punched cards. The update cards are processed by a routine which finds the designated pin or card, makes the indicated change, prepares a new file, and causes a replacement sheet to be printed showing the new conditions.

The EPL is used not only for the subsequent preparation of wiring data and schematics but its format (inputs grouped with outputs) also permits the generation of load lists which are automatically checked for overload. Such checking is prudent after a series of manual updates may have thrown a source over the allowable loading limit. In addition, load lists are added to logic schematics derived from the EPL.

Generation of Information for Interconnecting Circuit Cards
(See Fig. 1d.)

The Element Pin List represents a collection of cards which contain the circuits necessary to perform the computer functions. Some of the interconnections are already made on the cards, but the cards themselves must be interconnected. This is accomplished by a combination of etched back panels into which the cards are plugged, and harness wiring to interconnect the etched back panels. The program prepares punched tape information to control automatic machinery which produces the etched back panels, and it prepares wiring lists from which the panels are manually interconnected.

As the first step of this process, the program must determine which signals extend beyond the confines of a single etched panel so that suitable panel interconnection points may be provided and connected by etched wiring. These connection points are not specifically called out at this stage because the specific points will be selected by the etched layout program.

These general connection points, together with the other pins included within an etched panel, are processed in a group. Each pin in the group is associated with a particular signal. Pins bearing the same signal are wired together. As the layout proceeds, specific pins connecting to other panels are selected and listed separately for use by the harness wiring program.

The etched panel permits connections through horizontal etched copper strips and through vertical jumper wires inserted into holes in the panel. Thus, a typical connection between two pins may consist of a series of horizontal and vertical segments zig-zagging across the panel. Isolation of one run from other runs is accomplished by means of breaks in the copper strips which are provided in the etching process.

The program must seek an open path - there is a limited number of vertical and horizontal lines available - by testing for cuts and for existing jumpers which bar the way. The trials are made and a path found much as a rat finds its

way through a maze, except that certain preferences and patterns are established to guide the process. In addition, the task is somewhat simplified by seeking merely a satisfactory rather than an optimum layout.

Once the layout is complete, the data is converted to x-y coordinates for the location of cuts in the copper strips and for the location of holes in the panels through which the jumpers will be placed and dip-soldered to the etched copper. A printed image also is prepared of the layout to show the location of holes, cuts, and jumpers for manual use. Commonly the program is unable to complete all the connections with its somewhat limited imagination and rules; uncompleted connections are listed for manual completion with the aid of the printed image of the panel.

The harness pins are collected until all the etched panels are layed out. Again, pins bearing the same signal are wired together. The routing is done to reasonably minimize the length of each string of wires. The resulting list of "from-to" connections is grouped to provide convenience to the manufacturing operation. The connection items are also sorted on signal name to provide a signal reference list. Another list is provided by reversing each connection item, adding it to the list, and sorting the combined data on the "from" pin number to produce a pin reference list. And finally, the wiring strings are sorted according to the lowest pin number in each string to produce a list which allows semi-automatic checking of the wiring by ringing out each string in orderly, non-redundent succession.

Minor changes are made manually on the etched layout data through use of the layout image. Changes to the harness are made by preparing add-delete punched cards for each wire added or deleted. These cause the program to update the list and produce a replacement sheet for the pages thereby affected.

Preparation of Logic Schematic Diagrams (See Fig. 1e.)

These automatic processes which convert logic equations into lists of circuit cards and their interconnections must also report what they have done. The chief report is in the form of logic schematic diagrams. (Sample in Fig. 6.) These diagrams show in graphic form how the elements are interconnected in logical patterns to enable the user to trace rapidly through a series of elements and to identify the function of each element readily. Input and output pin numbers are shown. Names and pin numbers of loads are included.

These diagrams are generated from the Element Pin List; as a result, the diagrams match the wiring information because both are derived from the same source.

The logic schematic program involves a certain amount of pregrouping of the information through sorting and editing operations. The main format routine traces through the data in reverse order (upstream) to find and lay down the elements in the print image. This process also serves as a useful check on the completeness of the data, for should an element be deleted in error, its absence would break the chain and cause other elements to be "left over" at the end of the tracing process.

The important task of keeping the documentation of a computer in accurate and complete condition is aided, not only by the common-source generation referred to above, but also by use of an automatic updating process. When a change is made to the Element Pin List, the schematic program automatically detects such a change and produces a series of operations which finally result in printing a set of replacement sheets for all those previous sheets which experienced any changes.

All the output documents which are to be distributed to various using groups are printed on continuous-feed, preprinted vellum forms. A double-faced carbon sheet imprints on the back of the vellum and produces a carbon copy for immediate use while the "tracings" are being reproduced. The various document producing routines completely title, number, and otherwise identify the drawings according to standard practice.

All the output data is printed on a conventional printer having the usual scientific characters such as parenthesis, equal sign, and such.

General Comments

The programs consist of over 60,000 single address instructions for the IBM 704 Computer and are produced through regular assembly procedures (such as the SHARE Assembly Program). They were

developed jointly by engineering and programming groups within the Computer Department. Communication from the engineers to the programmers was primarily by means of conventional flow diagrams augmented by prose specifications and tabular control data for specific situations.

Use of the Automation Program as a Design Tool

One of the first ways that a logician can record his abstract ideas is in the form of an operation flow chart. This chart lists the micro-operations and their time sequence within a given machine instruction or command. An automation program gathers all the micro-operation and timing information and produces a concise set of organized documents called "Block Descriptions". The Block Descriptions (sample shown in Fig. 2) illustrate the control equations for every micro-instruction interval of time in all machine instructions. The logician combines the equations from each of the Block Description time intervals to form the complete Boolean equation for each "stage" of the proposed computer. (A proposed computer is sub-divided into a series of registers and a register is further subdivided into stages. A "stage" is a logical device the fan-out of which can be greater than one, for example: a single driver, or a single flip-flop.) The summation of equations for all stages of the computer then form the complete logical description of the proposed computer. For a moderately large scale computer this may result in 1000 Boolean equations for 1000 stages. Each equation describes the logical operations and connections to be made within a stage. All the equations, as a group, describe the interconnections to be made among stages of the proposed computer.

The Boolean equations now form the input data to a group of computer programs which generate manufacturing information for the factory. When the equations are written, it is assumed that all named logic signals are capable of driving an unlimited number of loads. Later programs either will remedy overload conditions automatically or complain to the logician.

The equations are punched into cards. The punched card format is entirely variable (see Fig. 3), meaning that equations are punched on the cards with few restrictions. The philosophy is to orient the input data to the logician and leave the interpretation problem to the computer program. (Refer to Fig. 3 for a sample equation listing.)

The Element Input List program that interprets the equations has three functions. These are:

1. To conduct exhaustive checks of the hardware implication of the equations for circuit rule violations.
2. To generate what we call an Element Input List. (As defined previously an "element" is the smallest subdivision of a computer which still retains its logical identity, e.g., AND gate, OR gate, flip-flops, etc.)
3. To assign all loads to appropriate sources within each stage and make a load list.

The "Element Input List" is a list of the elements needed to implement the proposed computer. These elements have been named and their input signals have been specified by logic name. After the Element Input List is generated, all the loads implied by this list are assigned to sources within stages. Quite often, this means the program must insert power drivers to relieve overload conditions. After the loads are assigned a load list is made that shows the loads, including wiring capacitance, presented to every source in the machine.

From this first series of programs which generate the Element Input List, there are a tremendous number of possible error print-outs which the logician may receive. Examples of possible error print-outs are:

1. Gates too wide.
2. Depth of logic too great.

How many of these error print-outs occur is a function of how well the equations conform to the circuit rules. (Refer to Fig. 4 for an example of error print-outs.) When an error print-out is made, the logician can either modify his equations to remedy the trouble, or, if he can show that special conditions exist, it may be possible to ignore the print-out.

The next step in the design of the proposed computer is to assign the hardware implied by the logic equations to printed circuit cards and locate these cards in the computer cabinets. When an Element Input List acceptable to the logician has been produced, he can proceed to the group of programs called "Card Assignment" that will assign the

implied hardware and locate it in the computer cabinets.

The purpose of this set of programs is as follows:

1. To generate an Element Pin List; and
2. To physically locate the plug-in cards within the computer cabinets.

The Element Pin List is permanently maintained on magnetic tape and is printed for the logician to examine. The printed Element Pin List displays one plug-in card on each page of paper. (Refer to Fig. 5 for example of Element Pin List.) The Element Pin List is also accessible to the logician for update purposes. Generally, updating of this list is for purposes of either optimizing the program results or for changing the computer's logic.

The logician's gross control of the physical location of plug-in cards within the computer is achieved through the use of the Module-Register table. This table shows either the number of stages from each register to be put into a module; or the number of plug-in cards of each register which are allocated to each module. He is aided in making this table by a program which prepares the plug-in card count necessary for each register. A "module" is a group of up to 28 plug-in cards which go together into one area of a computer cabinet.

There is a large amount of checking within these programs to see that the resulting Element Pin List conforms to certain requirements and restrictions. Some of the items being checked are:

1. An insufficient number of plug-in card positions allocated to a given register;
2. Clock omitted from flip-flop; and
3. Register not assigned a location in the computer.

If a rule violation is uncovered, there is an on-line print made (as was the case in the Element Input List programs). The logician then may remedy the violation by one of the following three methods:

1. Modify the input equations and re-run all affected data;
2. Make the modification manually by updating the element pin list; or
3. Show that special conditions exist which permit the apparent violation.

Logic schematics are made from the Element Pin List. (Refer to Fig. 6 for example of machine produced logic schematic.) These schematics are made entirely by machine programs and are printed on a conventional line printer. Each schematic shows the logic interconnections and physical connections (pins) within each stage of the computer. The schematic drawings are organized by stages and listed in sorted order by logic name. A load list is presented on the schematic which indicates the logic name and pin number of each load on the stage in question.

There is checking in the schematic program for a "closed system" of elements. Examples of some of the checks include:

1. Logic element called for (as an input signal) does not exist in Element Pin List.
2. Extra logic element was named that was never used.

Errors uncovered by the programs can be corrected either by updating the Element Pin List, or by modifying the input equations and re-running those portions of the computer design data that are affected.

From the Element Pin List, the wiring information for manufacture is generated by use of the wiring programs. Our particular "product design" is one that is wired with semi-automatic machine tools. In particular, the modules (with 28 plug-in card positions) are wired with printed circuit back panels. Horizontal "wires" on the back panels are of etched copper and vertical insulated jumper wires are put in place manually. The harness which interconnects all the module back panels is made manually. The control data for the automatic machine tool are in the form of two punched paper tapes for each module. One tape is used to operate an automatic drilling machine to drill pin holes and jumper wire holes in the back panel. The other tape is used to operate the same machine tool, but a light source is substituted for the drill. The light exposes a negative which controls the back panel etching. Data on both of these punched paper tapes is in the form of X - Y coordinates to position the machine tool table.

During the operation of the wiring program each of the module back panels has all the vertical and horizontal "wires" determined. An image of the back panel is printed out for visual check and reference before manufacture.

The wiring programs also produce four printed lists. The printed lists describe the harness which interconnects the back panels.

The first of these printed lists is the "Logic Sorted Pin List" as shown in Fig. 8. All the pins are arranged in the order of their logic name. This Logic Sorted Pin List is used as a reference document for maintenance of the computer.

The next printed list is called the master pin list. This list is sorted by pin number and shows the connections to each pin in the harness. During manufacture this list serves to check that the proper number of wires are connected to each harness pin.

The third printed list is the manufacturing list. This is a "from-to" list of wires. It is sorted on logic name but is partitioned to match the manufacturing assembly process.

The fourth and final printed list is the checking list. Semi-automatic equipment is used to check that listed harness pins have been connected together during manufacture, and this list serves as the input checking data for the semi-automatic equipment.

After a machine has been designed and sent out into the field, these same program-produced documents comprise much of the documentation for the computer. The Block Description, Logic Schematic and Logic Sorted Pin List are the three main documents sent into the field.

While this completes the description of the design of a computer using design automation programs, it might be of interest in retrospect, to observe the philosophical approach taken in the programs, and to make certain comments on the logic nomenclature used in the input data.

From the beginning, it is always assumed that the manual input data is in error. Therefore, the programs continually check the data for format, and circuit rule adherence.

The initial run through each major program usually results in several pages of on-line error prints for a moderately large scale computer design. It is always hoped that the programs have been designed flexibly enough to handle all the varied configurations the logician can dream up. It is not unusual that the first pass through a program with new data will uncover data configurations that "we would never use," (so said the logician) but which would indeed cause the program to halt. In such instances, a small amount of repair to the program is all that is necessary before it is running again.

When automation programs are made available to the logician, it becomes vital that a rigorously descriptive logic nomenclature be used. Stated in another fashion, the only variable data supplied to the automation program is the input logic data. If this data partially or clumsily specifies the proposed computer - other information being implied - the automation programs are unnecessarily made more complex.

The logic nomenclature system we are using progressively subdivides a computer. The computer is subdivided into registers. The registers are subdivided into stages. The stages are subdivided into elements. The name of a signal produced by an element is identical to the name of the element. Every logic signal within the computer will be associated with its own stage and register by the logic name.

The logic name is a 10-character fixed field as shown in Fig. 9. The alphabetic characters in the first four characters identify the register from which the logic signal in question originates. The first six characters - in total - identify the stage which originates the logic signal. The remaining four characters identify and describe the specific element within the stage.

Our pin number nomenclature, as shown in Fig. 10, provides a similar partitioning so that the complete pin number identifies the physical location of that pin.

Use of the Automation Program Output Documents

Three documents of primary importance emerge from the computer design phase.

1. Block Descriptions;
2. Logic Schematics; and
3. The Logic Sorted Pin List.

Moreover, after a machine has been designed and sent out into the field, these same program-produced documents serve as the total documentation for the computer.

When a fault is to be located in a computer, the first step is to determine the machine instruction in which the fault occurs. This isolates trouble shooting to a few pages in the Block Description. Tests are made until a faulty stage signal has been isolated within a register. Then the trouble shooter moves on to the Logic Schematic. With the Logic Schematic, the trouble can be isolated to one or two wires, or it may be located. If more information is needed, the trouble shooter refers to the Logic Sorted Pin List to determine the exact routing of the wires in question. Note that trouble shooting proceeds in an orderly fashion through the documents. There is a minimum of "page flipping" and of moving from one document to another.

While the output documents have been organized to allow orderly trouble shooting, the most outstanding advantage of these machine-produced documents is not their ease of use but their ACCURACY. One can be absolutely certain that all three documents exactly reflect the hardware being maintained and that they are consistent within themselves. Also, the continuity of information from one document to another, through the logic name, is assured since all three documents basically originate from the equation input data.

Although these three documents have been briefly mentioned previously, it is now necessary to explain them in more detail.

The Block Description is a sheet with a standard format that shows all registers, error flip-flops, control flip-flops, and the like that are basic to the computer control. For each of these registers, flip-flops, and such there is a blank space to be filled out which shows the control signal that is activated to put the registers and flip-flops into operation. If this control space is left blank, it means that the flip-flop cannot be operative during this micro-step. Every micro-step of every machine instruction has a Block Description sheet showing appropriate control signals. It is typical to have ten or twenty Block Description sheets for one machine instruction. (Refer to Fig. 2 for example.)

For trouble shooting, it is usually known which machine instruction is failing. This isolates the trouble to ten to twenty sheets of the Block Description. Next the micro-step that is failing is located which isolates the trouble to one sheet of the Block Description. Note that this one sheet gives all the pertinent information on the internal state of the computer.

The Logic Schematic displays in pictorial form the structure of the logic networks. Also, sufficient information is included so that equivalent points in the hardware can be located easily for testing. (Refer to Fig. 6.) The entire set of logic schematics for a machine forms a "closed loop" type of document. This means that if you start at some place in the drawings and trace downstream from that point you will eventually be able to trace back to the starting point from the upstream side. Thus, for trouble shooting it is necessary to be able to start at any point in the schematic and trace upstream or downstream with ease. Our schematics are organized by stages and are arranged in sorted order by stage logic names.

Upstream tracing with these schematics is done by extracting the logic name which serves as an input to the stage in question and proceeding to this stage of the logic schematic. Downstream tracing is easily accomplished by using the load list included on each stage schematic. While the load list gives all the downstream load information, the load stage schematic also may be located by the load logic name if necessary.

Due to its simplicity, the Logic Sorted Pin List (see Fig. 8) mentioned previously will not be described further except to say that the logic names used in the Block Descriptions and Logic Schematics are also carried through to the Logic Sorted Pin List.

Automation Program Effects Upon the Human Organization Designing and Building Computers

There are about 60 separate routines which together make up the five major programs we have described. These routines are largely stored on a program magnetic tape, although some are stored on punched cards. All of these routines were written for use on an IBM 704 Computer. The machine operator must select the routine and routine sequence he wishes to use on the particular input data. Generally these decisions are made before "getting on" the machine, and a deck of control cards provides the sequence of routines to be called in from the program

tape.

Generally, the machine operator function is performed by one person whose sole task is to run the programs on the data supplied by the logicians. Many technical decisions are made while running the routines. Quite often, the on-line error printouts from the programs will dictate immediate termination of a machine run until corrections in the data can be made. Sometimes error printouts may be ignored. Therefore, the operator has to be an engineer selected from the staff of computer design engineers. This machine operator, however, spends only a moderate portion of his time on the machine. The remainder is involved in organizing data to be run, keeping accurate control on the large number of magnetic tapes which represent the various machines being designed, and communicating with the logicians to give them results or to show them where the input data is faulty.

The logicians are responsible for preparing the input data for their particular machine design. Usually, this is accomplished by recording the data on forms and having the cards punched at one central location. Then, it is the logician's task to check the listing of these cards and to order that certain machine runs be made on this data. The machine run order request is made to the machine operator. The operator then schedules the machine time for the run, makes the run, and returns the results to the logician.

The basic input data to these programs is the Boolean equation representation of a proposed computer. Any equipment that is described with equations can be processed by these same programs. Therefore, the programs are invariant to the logic of the machine being designed. This is borne out by the fact that eight separate and distinct computer equipments (systems or parts of a system) have been designed to date with the same programs.

It has been indicated that a moderately large scale computer would have equations for about 1000 stages. These 1000 equations are the prime input data to this automation process. For the NCR 304 Central Processor which was built for the National Cash Register Company by the General Electric Company, there were about 1000 equations that were stored on approximately 2000 punched cards.

Listed below are some of the running times of these programs on a 704 Computer. These running times were for the design of the Central Processor portion of the NCR 304 Computer.

Program	Initial Run	Update Run
Block Description	6 hrs.	*
Element Input and Load Lists	3-1/2 hrs.	1 hr.
Card Assignment	3 hrs.	*
Logic Schematic	2 hrs.	1 hr.
Wiring	8 hrs.	*

*Approximately equal to the update volume divided by the total data volume times the initial running time.

As illustrated, a complete design can be done rapidly, assuming perfect input data. With this ability to generate new designs easily and with speed, the automation programs now can assume an entirely new role. Such programs can be used as evaluators to test the effect of varying parameters and circuit rules to optimalize the design.

One excellent example of the evaluation role of the programs took place during the design of the NCR 304. A power driver plug-in card had been built and was within a few weeks of going into production for the NCR 304, when a transistor manufacturer announced a new transistor with about 70% greater load driving capability. However, the price of the new transistor was considerably higher than the transistor that might be replaced. The questions to be answered were:

1. Can the higher cost of the new transistor be justified?
2. Can the new driver be included in the design before the computer release to the factory in the immediate future?

With the adjustment of a few constants within the program and a few hours machine running time, it was simple to justify the higher priced transistors and generate all the new data required for the factory.

To summarize the effects these automation programs have had upon the Engineering organization using them, here are some of the immediate and rather obvious results which were achieved:

1. Reduced computer design cycle time;
2. Reduced engineering design manpower;
3. Better documentation of the

designed computer;

4. Greater reliability built into the computer through complete adherence to circuit rules;

5. Better optimized computer design; and

6. Drastic reduction in the educational effort necessary to teach the logicians the circuit rules.

Now, let us examine some of the above items in detail. An engineering model of the NCR 304 Central Processor was built by entirely manual methods. The production model NCR 304 Central Processor was built by using the automation programs. The production model bore no physical resemblance to the engineering model. Even the plug-in cards were completely re-designed. Thus, both designs were different except for the logic, and both designs started with approximately the same logic equations.

Considerably less engineering manpower was needed for the design of the production model than the engineering model even though the production model was a much more complete design. The non-engineering personnel remained about constant during the two designs. Using the same manpower on the automated design as on the manual design, the computer design cycle could be reduced, and here it should be noted that with the automated design, a few key engineering people can run the entire design effort rather than spreading the design among many groups of engineers. Stated in another fashion, the design task has been modified such that non-engineering personnel can do much of the work formerly done by engineers, and yet a few engineers can maintain absolute control of the whole process.

One reason why fewer engineering man-months are required in the automated design is that most of the "engineering" decisions of how to implement logic with hardware are made by the programs. In addition, having the programs implement the logic into circuits produces an additional effect of great importance. This is that ALL logic configurations are thoroughly checked to conform to circuit rules. Such things as wire capacitance loading on every source are accurately calculated and factored into the total loading picture. With an entirely manual design process, this would be almost impossible even though reliable machine operation dictates such calculations.

With the implementation of logic into circuits by program, the designer only needs to know the circuit rules in a gross sense. The circuit designers develop the circuit and its rules and deliver this detail information to the programming group who builds the rules into the program. There are a total of only two or three people in the programming group who ever see these detail circuit rules as opposed to educating a large number of personnel as was formerly required. Thus, it is evident that one of the difficult and important communication problems in digital computer design has been circumvented.

When the Design Automation techniques were introduced, there was considerable resistance to them in certain areas. Familiarization with the new documents, however, brought firm support for the new documents and their many benefits in a very short time. The greatest advantage, perhaps is the knowledge that the documents always exactly represent the equipment in question. Also, the machine updating of the drawings results in a very short time lag until changes are reflected in the drawings.

The digital computer design task is never static. It will be continuously and progressively automated by the use of programs run on existing computers. Better optimized designs and design processes in which such matters as accuracy and adherence to circuit rules are effectively handled, both serve as stimuli for automating the design task.

Acknowledgements

The authors of this paper wish to express their appreciation to the following employees of the Applications Section of the General Electric Computer Department for their efforts and cooperation in writing and debugging the Design Automation Programs:

M. A. Anfenson
R. B. Cochran
D. D. Degler
W. T. Dodds
P. H. Jennings
R. W. Miller
R. E. Moore
W. J. Willis

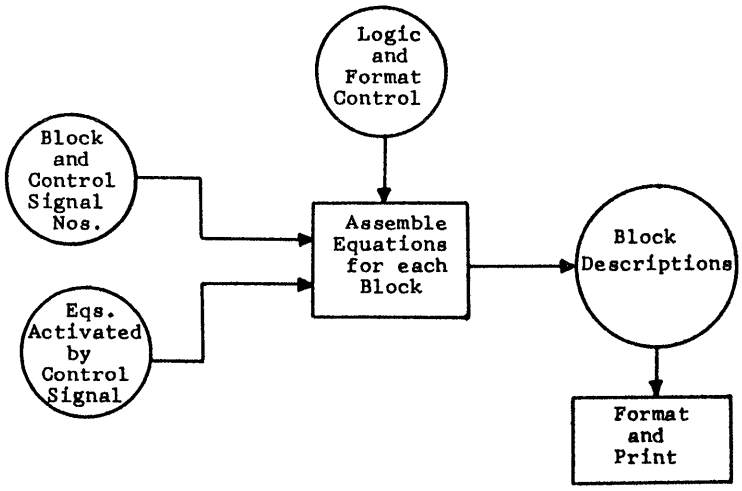


Fig. 1a. Over-all Design Automation Flow Chart

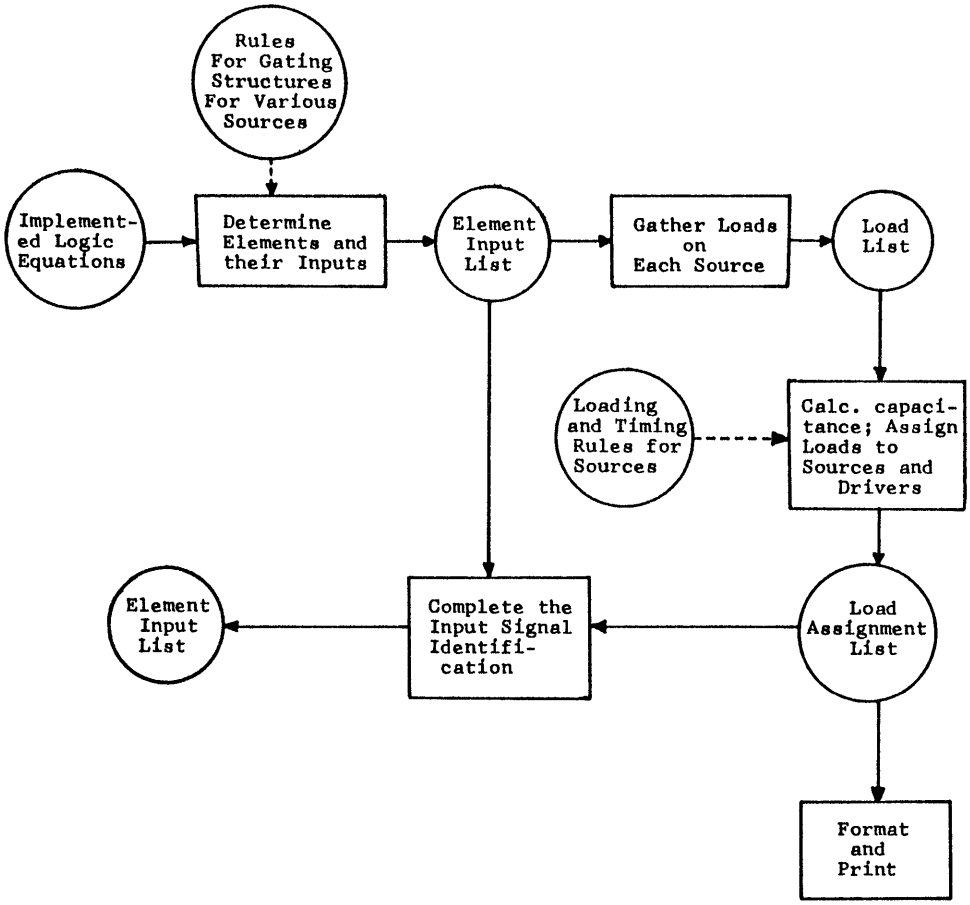


Fig. 1b. Over-all Design Automation Flow Chart

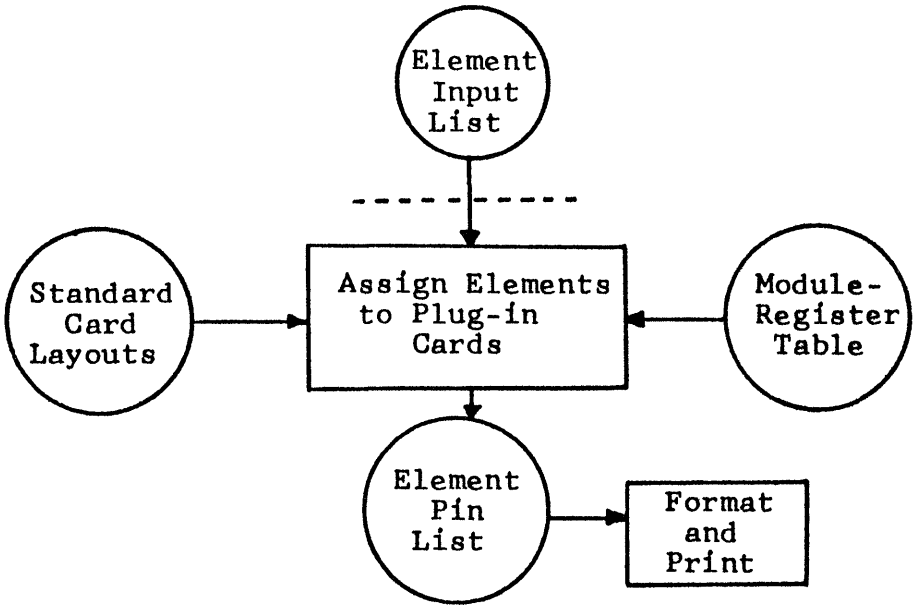


Fig. 1c. Over-all Design Automation Flow Chart

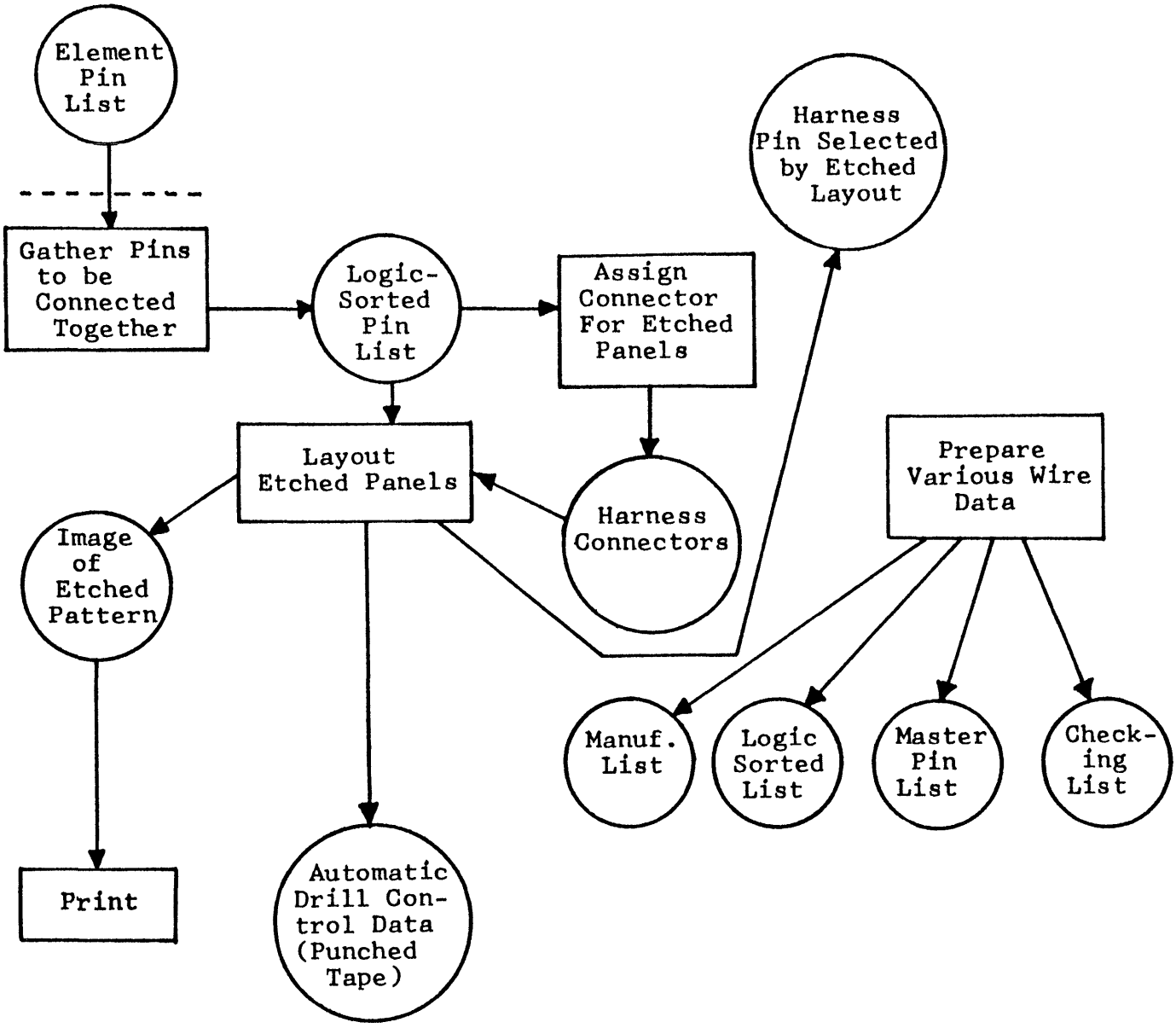


Fig. 1d. Over-all Design Automation Flow Chart

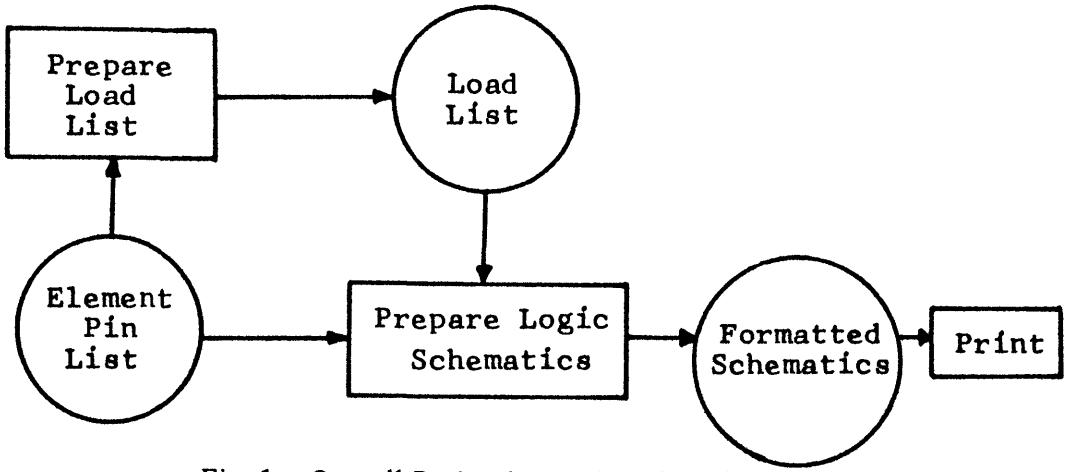


Fig. 1e. Over-all Design Automation Flow Chart

Fig. 2. Sample page from Block Description

Figure 2 is a representative Block Description sheet showing the internal state of the machine for one micro-instruction—commonly called a block. The illustrated Block is the second micro-instruction (02) for the ADD command (instruction #01). The right side of the drawing is the prose description of the block. It can be seen that some of the equations have been left blank to signify that those flip-flops, etc., are not operative in this block.

```
     BLOCK NUMBER      0102 0

-------------------------------------
ADDRESS REGISTER  (L-REG)

FL011-FL041 = FA011-FA041 XP141 PL071
FL010-FL040 = FA010-FA040 XP141 PL071
FL051-FL141 =
FL050-FL140 =             XP141 PL071
-------------------------------------
A AND L REGISTER TIMING

FL001 = XP051 PL251
FL000 = XP081
-------------------------------------
AUX.ADDRESS REGISTER  (A-REG)

FA011-FA081 = FA051-FA121 FL001 XA051
FA010-FA080 = FA050-FA120 FL001 XA051
FA091-FA121 = FM011-FM041 FL001 PA021
FA090-FA120 = FM010-FM040 FL001 PA021
FA131       = FM051 FP281 FL001 PA021
FA130       = FM050 FP281 FL001 PA021
FA141       = FM061 FP281 FL001 PA021
FA140       = FM060 FP281 FL001 PA021
-------------------------------------
ARITHMETIC UNIT

XF011-XF061 = FS011-FS061 XFA11
XF010-XF060 = FS010-FS060 XFA11

XG011-XG061 = FM011-FM061 XGA11
XG010-XG060 = FM010-FM060 XGA11

FKA11 = NJ171 FP001 XKA11
FKA10 = NJ170 FP001 XKA11
FKA10 = FP141 XKA90

DECIMAL  ADD
-------------------------------------
MEMORY REGISTER

NO CHANGE OF CONTENT IN LOGIC PERIOD

MEMORY-SELECT = ADDRESS IN L-REG.
-------------------------------------
SPECIAL STORAGE REGISTER  (S-REG)

FS111-FS121 =
FS110-FS120 = FS001 XS131

FS131-FS141 =
FS130-FS140 = FS001 XS141

FS151       =
FS150       = FS001 XS151

FS161       =
FS160       = FS001 XS161

FS001 = XP231 PS431
FS000 = XP091
-------------------------------------
ADDRESS COUNTER    (DS-COUNTER)

FD011-FD041 =
FD010-FD040 = XP201 XD001

FD051       = XP201 PD081
FD050       =
-------------------------------------
EXTRACTION REGISTER  (E-REG)

        NOT COUNTING

EE211-EE241 = FM011-FM041 FE001 PE021
EE210-EE240 = FM010-FM040 FE001 PE021

FE011-EE201 = FE051-EE241 FE001 PE041
FE010-EE200 = FE050-EE240 FE001 PE041
FE001 = XP231 PE231
FE000 = XP051 PE351
FE000 = XP221
-------------------------------------
TALLY REGISTER  (B-REG)

    NO CHANGE
-------------------------------------
MISC. BIT STORAGE

FKM11 = (FL01-14 = CONSOLE SWITCHES)
        JI401 XP231 (M-MAIN)
FKM11 =(JI413K + JI423K(FM011+FM021)+
        JI433K FM021+ JI443K FM011
        FM021) XP081 PN231
-------------------------------------
FKOR1 = XP08 FM051 PN231
-------------------------------------
DECISION LOGIC

FK011 =

FK010 = XP141

FK021 = (FA05-07= 000) XP101 PN231

FK020 = XP141
-------------------------------------
INSTRUCTION REGISTER

FN141-FN161 = FM011-FM031 SR031 XP091
              PN231
FN140-FN160 = FM010-FM030 SR031 XP091
              PN231
FN171       = FM051 SR031 XP091 PN231
FN170       = FM050 SR031 XP091 PN231
-------------------------------------
PROGRAM COUNTER  (FN01-FN07)

  FK010 FK020 - COUNT TO NEXT BLOCK

  FK010 FK021 -  SKIP TO BLOCK    04
        PN031

  FK011 FK021 -  SKIP TO BLOCK    00

  FK011 FK020 -  SKIP TO BLOCK    00
-------------------------------------
I
I COMMAND-   ADD
I
I BLOCK-  02     FN14   FN17
-------------------------------------
LOOK UP SECOND WORD OF
COMMAND
-------------------------------------
```

The automonitor FF is set if the monitor address selection switches are equal to the L-reg. and the monitor address switch is on; or if the automonitor character is equal to or greater than the monitor character selection switch setting.

The B & C portions of the second word of the command, containing the partial word designations, are copied into the E-reg. from the M-reg.

The S2-reg. is cleared.

The A portion, which contains the monitor character and relative addressing instructions, are copied into the A-reg.

If the monitor character is negative, the over-ride FF is set.

The mode bits of character 9 are copied into the N-reg. providing the machine is not in R3 test mode.

The index register character (R) is copied into the L.S. four bits of the L-reg. while the remainder of the L-reg. is cleared.

The program counter counts to the next block if the index register field selector (S) is not equal to zero.

The program counter skips to block 01-04 if the index register field selector (S) is zero.

```
EW110A = QC011A * FP201A NG010A *
EW111A = QC011A * PG013A *
EW120A = QC011A * EW121A FG040A *
EW121A = QC011A * EW100A EW120A FG001A FG011A FP000A DP230A EW111A *
FA010A = QC011A * FA050A DA051A + FL010A DA061A *
FA011A = QC011A * FA051A DA051A + FL011A DA061A *
FA020A = QC011A * FA060A DA051A + FL020A DA061A *
FA021A = QC011A * FA061A DA051A + FL021A DA061A *
FA030A = QC011A * FA070A DA051A + FL030A DA061A *
FA031A = QC011A * FA071A DA051A + FL031A DA061A *
FA040A = QC011A * FA080A DA051A + FL040A DA061A *
FA041A = QC011A * FA081A DA051A + FL041A DA061A *
FA050A = QC011A * FA090A DA051A + FL050A DA061A *
FA051A = QC011A * FA091A DA051A + FL051A DA061A *
FA060A = QC011A * FA100A DA051A + FL060A DA061A *
FA061A = QC011A * FA101A DA051A + FL061A DA061A *
FA070A = QC011A * FA110A DA051A + FL070A DA061A *
FA071A = QC011A * FA111A DA051A + FL071A DA061A *
FA080A = QC011A * FA120A DA051A + FL080A DA061A *
FA081A = QC011A * FA121A DA051A + FL081A DA061A *
FA090A = QC011A * FL010A DA011A + FM010A DA021A + NJ010A DA031A
           FS010A DA041A + FL090A DA061A + DA121A*
FA091A = QC011A * FL011A DA011A + FM011A DA021A + NJ011A DA031A
           FS011A DA041A + FL091A DA061A *
FA100A = QC011A * FL020A DA011A + FM020A DA021A + NJ020A DA031A
           FS020A DA041A + FL100A DA061A + DA121A *
FA101A = QC011A * FL021A DA011A + FM021A DA021A + NJ021A DA031A
           FS021A DA041A + FL101A DA061A *
FA110A = QC011A * FL030A DA011A + FM030A DA021A + NJ030A DA031A
           FS030A DA041A + FL110A DA061A + DA121A *
FA111A = QC011A * FL031A DA011A + FM031A DA021A + NJ031A DA031A
           FS031A DA041A + FL111A DA061A *
FA120A = QC011A * FL040A DA011A + FM040A DA021A + NJ040A DA031A
           FS040A DA041A + FL120A DA061A + DA121A *
FA121A = QC011A * FL041A DA011A + FM041A DA021A + NJ041A DA031A
           FS041A DA041A +FL121A  DA061A *
FA130A = QC011A * FL130A FP281A DA011A + FM050A FP281A DA021A + NJ050A
         FP281A DA031A + FS050A FP281A DA041A + FL130A DA061A
           FP281A DA121A *
FA131A = QC011A * FL131A FP281A DA011A + FM051A FP281A DA021A + NJ051A
         FP281A DA031A + FS051A FP281A DA041A + FL131A DA061A *
FA140A = QC011A * FL140A FP281A DA011A + FM060A FP281A DA021A
           NJ060A FP281A DA031A + FS060A FP281A DA041A
           FL140A DA061A + FP281A DA121A *
FA141A = QC011A * FL141A FP281A DA011A + FM061A FP281A DA021A + NJ061A
         FP281A DA031A + FS061A FP281A DA041A + FL141A DA061A *
FB000A = QC011A * DP001A PB503A + DP031A PB533A + DP041A PB543A
           DP071A PB573A + DP081A PB583A + DP091A PB593A + DP101A
         PB353A + DE321A FP001A PB643A + FS010A FS020A FS030A
         FS040A FS060A FP001A PB353A + DP011A PB513A + DP231A PB733A *
FB001A = QC011A * DP221A PB423A + DP231A PB433A + DP021A PB223A
           DP051A PB253A + DP081A PB283A + DE311A FP261A PB343A
           (FS011A + FS021A + FS031A + FS041A + FS061A) FP001A
         PB353A *
```

Fig. 3. Sample Input Equation Data

Figure 3 is a partial listing of the equations for a recently designed machine. The plus sign is an OR relation and the parenthesis is an AND relation. The logic signals are the six character groups. An AND relation is implied where no character appears between logic signals. The asterisk denotes the beginning and ending of the equation input terms.

```
SSW1 IS UP--OUTPUT IS ON TAPE 8
      DA051A      LEVEL TOO HIGH     DP071A000000000006
      DA051A      LEVEL TOO HIGH     DP061A000000000006
      DA051A      LEVEL TOO HIGH     DP041A000000000006
      DA051A      LEVEL TOO HIGH     DP031A000000000006
      DA081A      WIDTH ERROR IN GATE      DA081A5G011-
      DA081A      WIDTH ERROR IN GATE      DA081A4G011-
      DA081A      WIDTH ERROR IN GATE      DA081A3G011-
  ILLEGAL CHAR    EA021A
      EA081A      LEVEL TOO HIGH     DU011A000000000005
      EA081A      WIDTH ERROR IN GATE      EA081A3G011-
  FORMAT          EA801A        NO FIRST ASTERISK
  SEQUENCE        EA091A
  ILLEGAL CHAR    EA091A
  FORMAT          FA000A        CLOCK
      FA011A      CLAMP ERROR IN GATE      FA011A2G011-
      FA081A      WIDTH ERROR IN GATE      FA081A5G011-
      FA081A      WIDTH ERROR IN GATE      FA081A4G011-
      FA081A      WIDTH ERROR IN GATE      FA081A3G011-
      FA081A      CLAMP ERROR IN GATE      FA081A2G018-
  FORMAT          FA791A        NO FIRST ASTERISK
  FORMAT          FC010A       AD011AAD021A +
  FORMAT          FC010A        +     AD011AAD021A
  FORMAT          FC011A        NO FIRST ASTERISK
  FORMAT          FC021A        NO FIRST ASTERISK
  FORMAT          FC021A       DA045A
  FORMAT          FC021A       DA046A
  FORMAT          FC011A        NO FIRST ASTERISK
  SEQUENCE        FC011A
  FORMAT          EQ SUM       JC034A
      JC043A      WIDTH ERROR IN GATE      JC043A5G011-
      JC043A      WIDTH ERROR IN GATE      JC043A4G011-
  SEQUENCE        FN080A
  FORMAT          PA011A        NO  EF DJN



NO LOAD LIST FOR                      FP000A 0F01
NO LOAD LIST FOR                      FP000A 0F02
ELEMENT    FP100A       5G021-      DOES NOT APPEAR IN INPUT DATA
ELEMENT    FP100A       5G041-      DOES NOT APPEAR IN INPUT DATA
NO LOAD LIST FOR                      FP200A 0F02
TAPE 5 IS FULL.  REPLACE AND PUSH START.

NO LOAD LIST FOR                      FP240A 0F01
NO LOAD LIST FOR                      FP260A 0F01
NO LOAD LIST FOR                      FP290A 0F01
NO LOGIC FOR LOADS                    FP291A0N0A
ELEMENT    FS110A       4G071-      DOES NOT APPEAR, IN INPUT DATA
ELEMENT    FS120A       4G071-      DOES NOT APPEAR IN INPUT DATA
LEFT OVER ELEMENT       FS160A      4G01 X      03F6        1C2140
```

Fig. 4. Error Printouts from Element Input List and Logic Schematic Programs

In Fig. 4, the upper half of the example shows a number of error printouts made by the Equation Interpretation program. The lower half shows error printouts made by the Schematic program. Both examples are from Production Input data.

CARD LOCATION	CARD SERIAL NO. IN MOD.	CARD TYPE
01H1	23	04

CKT. TYPE	EQUATION SUM	EL.LEVEL & TYPE	PIN NO.	CKT. TYPE	EQUATION SUM	EL.LEVEL & TYPE	PIN NO.
2G2	DM771A	4G0111	47	4G2	DM801A	4G0212	08
	EM821A	0E020-	29		FL093A	0F01D-	17
	FL142A	0F01B-	28				18
2G2	DM771A	4G0111	45	4G2	DM801A	4G0312	09
	FL093A	0F01D-	41		FL142A	0F01B-	13
	FL103A	0F01C-	40				14
2G2			42	4G2	DM801A	4G0411	05
			37		FM810A	0F01E-	03
			36				04
2G2			44	4G2	DM801A	4G0511	06
			35		DM181A	0H013-	12
			34				10
2G2			46	4R			43
			31				
			32	4R			33
4G2	DM761A	4G0211	07	4R			21
	PM003A	0H0A0	19				
	EM821A	0E020-	20	4R			30
4G2	DM771A	4G0211	11	4R			38
	RM803A		15				
	EM821A	0E020-	16				

Fig. 5. Plug-In Gate Card in Element Pin List Format

Figure 5 is an Element Pin List representation of a single plug-in gate card. Note that some of the gates and some of the inputs have been left unused. Also, some pins have identical logic names attached–indicating that these pins will be wired together.

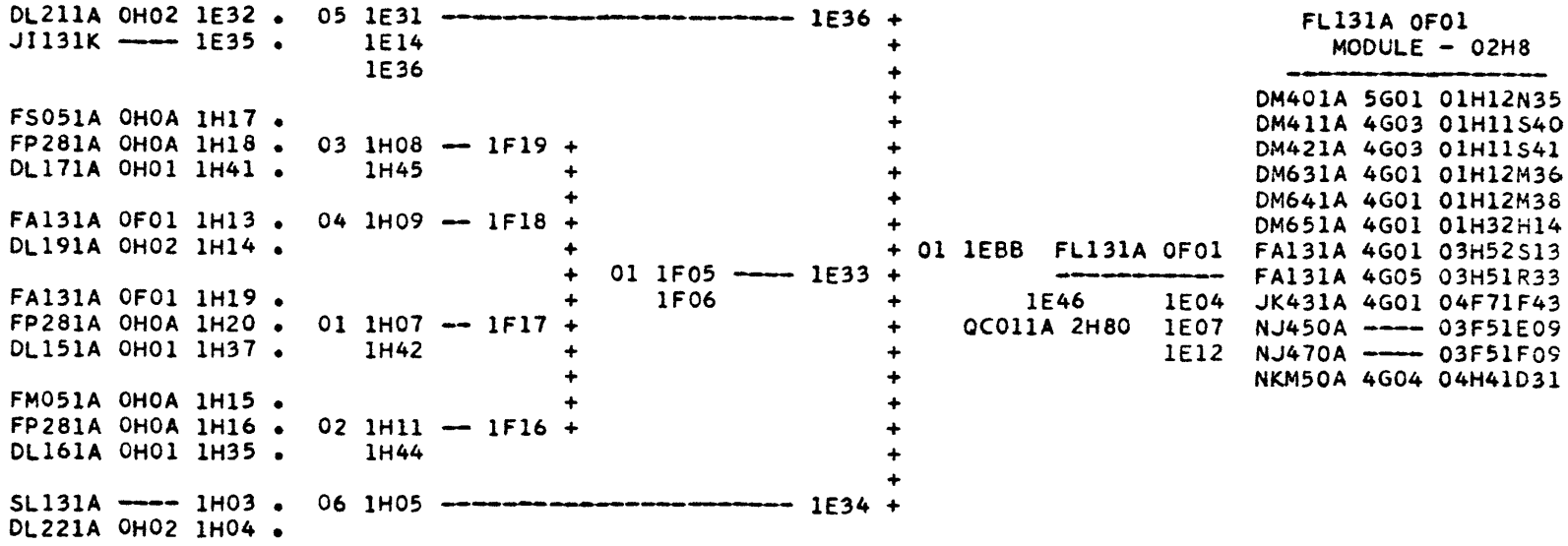


Fig. 6. Machine Produced Logic Schematic

The machine produced Logic Schematic displays the structure of the logic networks. In this schematic, interconnections between elements of the stage are made obvious by their placement on the drawing. Therefore, the use of interconnecting lines is unnecessary. Rows of periods designate the scope, or width, of an AND gate and rows of plus signs designate the scope of an OR gate. Any break in the sequence of these symbols terminates the scope of the gate in question. Input pins to elements are physically located with the four character pin number immediately to the left of the scope line for the element. To the right of the gate scope symbols is the two character gate name (serial number) and next to the two character gate name is a four character pin number identification of this gate output. If several circuits have to be connected in parallel to implement this gate, then several output pins will be present in a column. It is implied that all pins in the stage are contained in the cabinet, rack and module that is shown as the "module" in the upper right corner of the drawing.

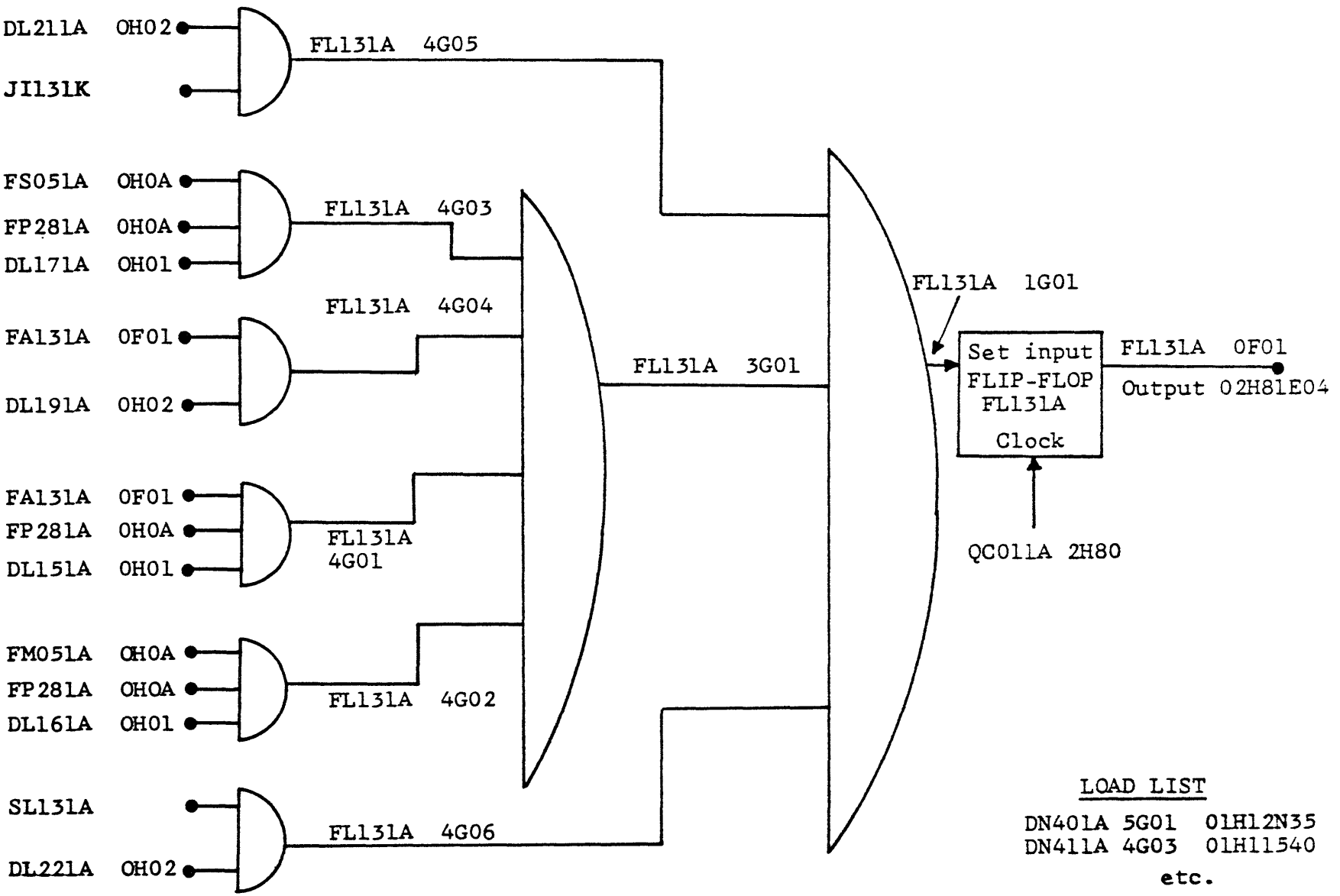
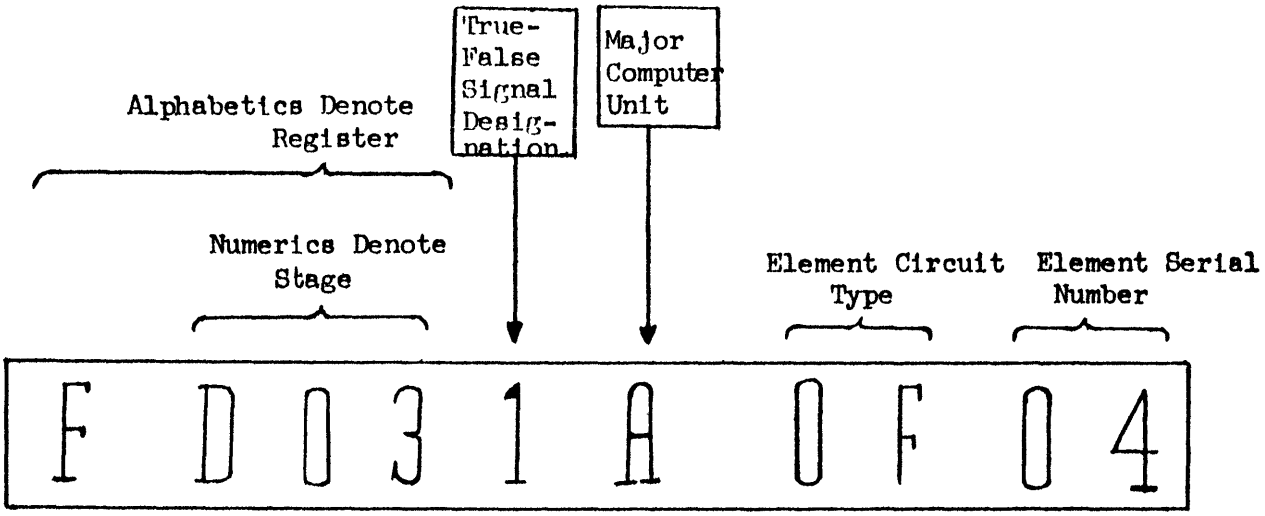


Fig. 7. Hand drawn Logic Schematic (with no pin information)
Figure 7 shows the equivalent hand drawn logic schematic for comparison to the machine produced schematic of Fig. 6. It should be noted that Fig. 7 includes no pin number (physical location) information whereas Fig. 6 has both logic and pin information.

LOGIC	PIN NO.
EN010E0E01	41H22M29
	41H32Z11
	41H41A38
EN011E0H0A	41H32Z39
	41H42A23
	41H51Z29
EN011E0H0B	41F19C07
	41H32Z05
	41H41Z33
EN020E0E01	41H41A11
	41H22M10
EN021E0H0A	41F31Z25
	41F41A40
	41H12Z40
	41H31Z03
	41H42A40
EN021E0H0B	41F19C06
	41F41A20
	41H52A20
	41H41Z32

Fig. 8. Sample page from Logic Pin List



The following information is carried in the above logic designation:

Third stage of the FD register

True signal

Central Processor logic

Flip-flop element - fourth serial number in stage

Fig. 9. Sample Logic Nomenclature

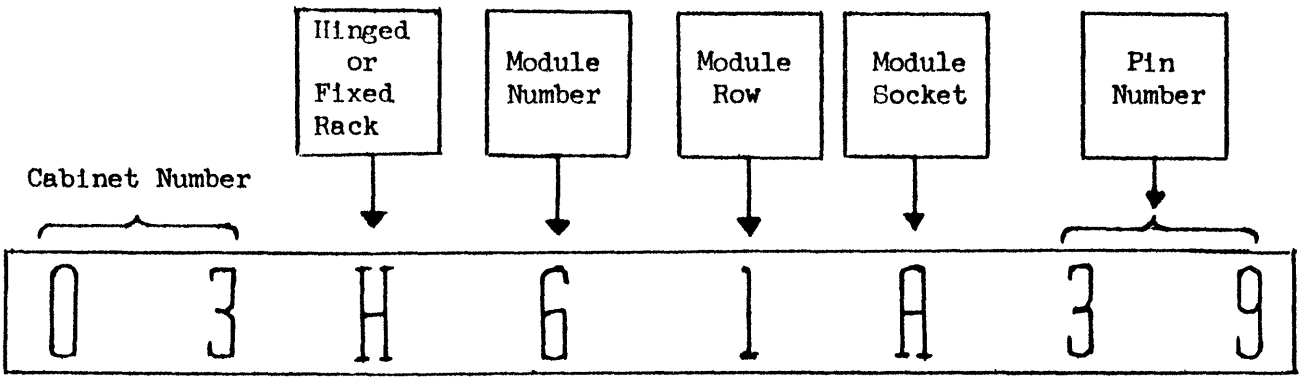


Fig. 10. Sample Pin Nomenclature

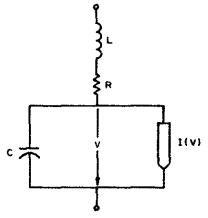
CALCULATED WAVEFORMS FOR THE TUNNEL DIODE LOCKED-PAIR CIRCUIT

D. R. Crosby H. R. Kaupp
Electronic Data Processing Division
Radio Corporation of America
Camden, N. J.

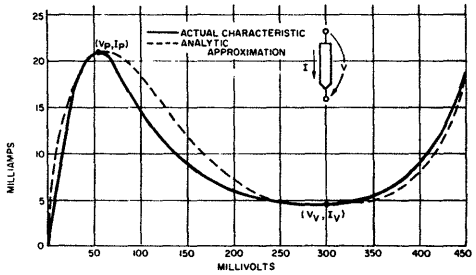
The purpose of this paper is to present an introductory analysis of the tunnel diode locked-pair circuit. The characteristics of the tunnel diode, together with the simplicity of the locked-pair circuit, make it a major contender for use as a high-speed computer element. [1, 2] High speed and high gain are the main advantages of the locked pair; the multi-phase power supply and lack of a simple means for logical inversion are the main disadvantages. The basic circuit consists of two tunnel diodes in series, the node common to the tunnel diodes being both the input and output terminal. As a computer element, the locked pair functions in much the same manner as the phase-locking harmonic oscillator (PLO). Like the PLO, the locked pair overcomes the difficulty of coincident input and output terminals by using a three-phase voltage source. However, the theory of three-phase majority logic is not essential to the understanding of this paper. [3, 4]

A. Basic Concepts of Locked-Pair Circuitry

The equivalent circuit of a tunnel diode is shown in Figure 1A, and the volt-amp characteristic of the nonlinear conductive element is shown in Figure 1B.



(A) EQUIVALENT CIRCUIT



(B) TYPICAL NEGATIVE CONDUCTANCE CHARACTERISTIC

FIGURE 1 - TUNNEL DIODE

To understand the basic operation of the locked pair, examine the idealized circuit of Figure 2, where diode inductance and capacity are neglected. Assume that tunnel diode B is the active element, and A is the load. For the moment, neglect the source resistance. Increasing the voltage E "draws" the load curve A across the characteristic of the active element B. This is analogous to what is done with tube and transistor circuits, the difference here being the nonlinear load. Figure 3 indicates the current-voltage relationships for the circuit of Figure 2, at a particular value of source voltage, where it is seen that the load curve A intersects the element curve B at three points. Points one and three are stable, while point two is unstable. The state in which the circuit locks, as the source voltage is increasing, depends on which of the two characteristics first reaches its negative conductance region. By inserting a locking current, I_L (Figure 2) the operating point is predetermined (Figure 4). The effect of I_L is to shift the family of operating points, causing element B to reach its negative conductance region before element A. Thus, diode B goes to its high voltage state. When the voltage E becomes equal to E_L, point three of Figure 3 will be the operating point, so that a locking current into the node causes the circuit of Figure 2 to have a high-voltage output. Conversely, a locking current out of the node would cause a low-voltage output (operating point 1 of Figure 3).

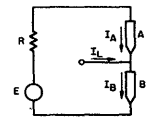


FIGURE 2 - IDEALIZED LOCKED-PAIR CIRCUIT (INDUCTANCE AND CAPACITY NEGLECTED)

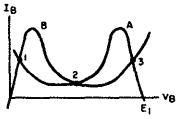


FIGURE 3 - CURRENT AND VOLTAGE RELATIONSHIPS FOR CIRCUIT OF FIGURE 2 FOR $E=E_I$

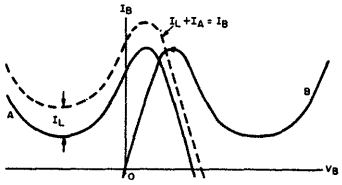
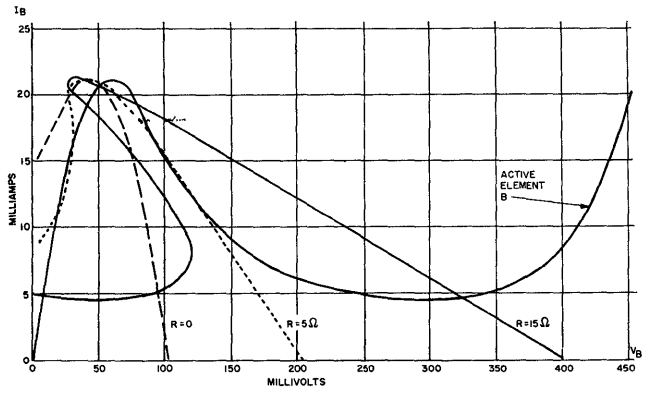


FIGURE 4 - RELATION OF CHARACTERISTICS NEAR CRITICAL POINT OF SWITCHING

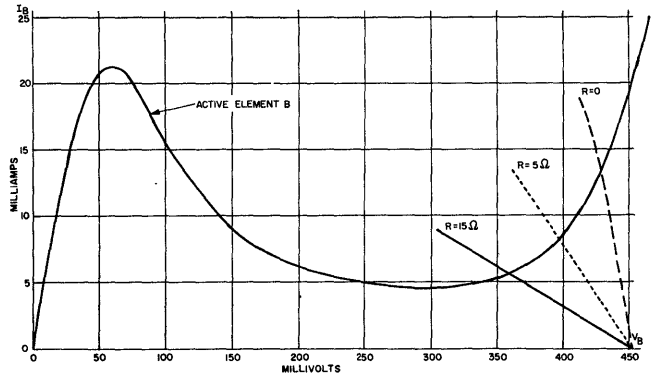
The locked pair is susceptible to a locking signal in the region where the peaks of the diode characteristic are crossing; thereafter it is relatively insensitive to spurious signals.

A significant difference in diode peak current produces the same effect as the locking current; hence, the locking current must be large enough to overcome the differences in diode characteristics.

The effect of source resistance on diode switching will now be examined. As in Figure 2, tunnel diode B will again be considered the active element. However, the load curve now is the series combination of the source resistance, R, and the V-I characteristic of diode A. The source resistance affects the peak of the load curve by moving it to a higher voltage. Hence, a larger source voltage is needed to bring the characteristics to the critical point of switching, which is indicated quantatively in Figure 5A. Figure 5B indicates the operating point of the circuit when the source voltage has reached an arbitrary maximum. The effects of a high source resistance are seen as a delay in switching, and a reduced output voltage.



(A) PRIOR TO SWITCHING



(B) AFTER SWITCHING

FIGURE 5- GRAPHICAL INTERPRETATION OF CIRCUIT IN FIGURE 2 FOR VARIOUS SOURCES RESISTANCES

The waveforms for the circuit of Figure 2 can be obtained by making a point-by-point plot from the characteristics shown in Figure 5. The current and output voltage waveforms in Figure 6 are plotted for a sinusoidal source voltage, and a source resistance of 5 ohms.

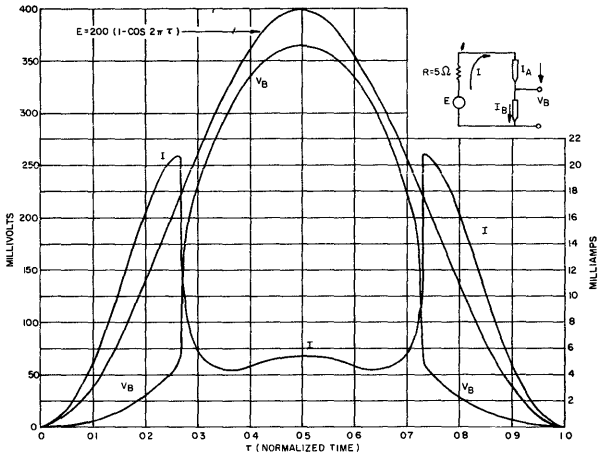


FIGURE 6-WAVEFORMS OF IDEALIZED LOCKED-PAIR CIRCUIT (INDUCTANCE AND CAPACITY NEGLECTED)

B. Computer Solutions

The technique related above and used for Figure 6 is invalid when the effects of capacitance and inductance are appreciable. Through use of a digital computer, a solution considering the effect of diode capacities, can be easily realized. Such a computer solution excludes the stray parameters associated with laboratory work at high frequencies, thereby disclosing the basic nature of the circuit.

For the ensuing discussion a sinusoidal source voltage is assumed, because it seems to be the most practical waveform for driving a large number of locked-pair circuits.

A d-c component is added to the sinusoidal source voltage to keep the diode voltage from having negative values. A negative voltage across the diode would cause an unnecessary loss of power, and if the diodes V-I characteristics are not well matched, the circuit would exhibit an undesirable output voltage during negative excursions of the driving voltage. Therefore, the source voltage will be of the following form as plotted in Figure 6:

$$E = K_1 - K_2 \text{ Cos } 2\pi \text{ ft}$$

The values of K_1 and K_2 are somewhat arbitrary. However, the d-c component, K_1, must be of such a value that the minimum excursion of driving voltage is less than the peak voltage of the diode characteristic (V_p in Figure 1B), otherwise the circuit will never relax. That is, the same diode will always go to its high state regardless of the polarity of the locking signal. For the problems solved on the computer, the chosen d-c voltage component was equal to the peak sinusoidal value:

$$K_1 = K_2$$

The nominal inductance of RCA germanium tunnel diodes is 0.4 muhenries, yielding an inductive reactance of 1 ohm at 400 mc. Thus, for the values of frequency and source resistance used in these examples, the inductance of the tunnel diode may be neglected.

RCA germanium tunnel diodes (nominal 20-ma. peak current) with identical nonlinear characteristics are assumed. To facilitate compution, the programmers** developed an analytical expression to approximate the nonlinear characteristic. The experimental tunnel diode curve is compared with its analytic approximation in Figure 1B.

The first problem was solved for the circuit of Figure 2 with diode capacity included. In order to keep the problem as simple as possible, no locking signal is applied; instead, the diodes switch because of the difference in capacity. In this instance, diode B always switches to its high-voltage state because it has the smaller capacity.

The defining equations derived from Figure 2 are:

$$E = K_1 - K_2 \operatorname{Cos} 2\pi\tau = IR + V_A + V_B$$

$$I = I_{1A} + I_{2A} = I_{1B} + I_{2B}$$

$$\frac{dV_A}{d\tau} = \frac{I_{2A}}{fC_A} \qquad \frac{dV_B}{d\tau} = \frac{I_{2B}}{fC_B}$$

where:

τ = ft, the normalized time (f = frequency in cps)

R = 5 ohms, the source resistance

K_1 = 0.2 volts, d-c component of source voltage

K_2 = 0.2 volts, peak value of sinusoidal component of source voltage

C_A = 170 $\mu\mu$f, capacity of tunnel diode A

C_B = 140 $\mu\mu$f, capacity of tunnel diode B

Solutions were obtained for frequencies of 10, 30, 100 and 300 mc. The resultant voltage and current waveforms are shown in Figures 7 through 10.

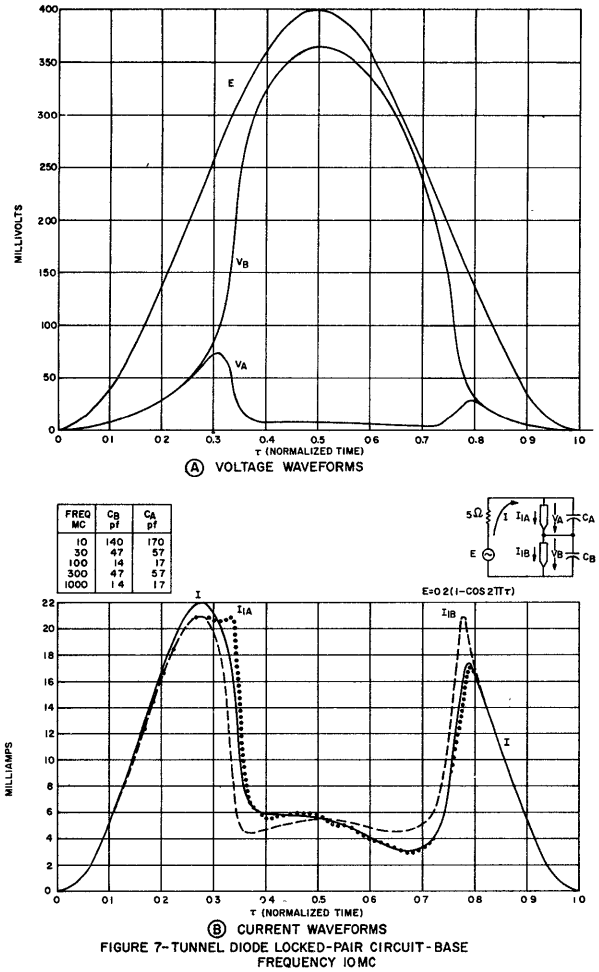


FIGURE 7- TUNNEL DIODE LOCKED-PAIR CIRCUIT - BASE FREQUENCY 10 MC

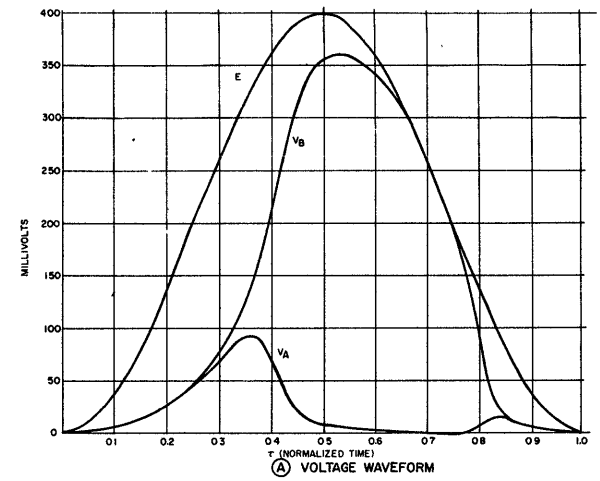
Note that capacity and frequency appear only as a product in the defining equations of the circuit. Therefore, the waveforms can be interpolated for different values of frequency and capacity through use of the relations:

$$f_{II} = \frac{140}{C_{BII}} f_I \qquad C_{AII} = \frac{170}{140} C_{BII}$$

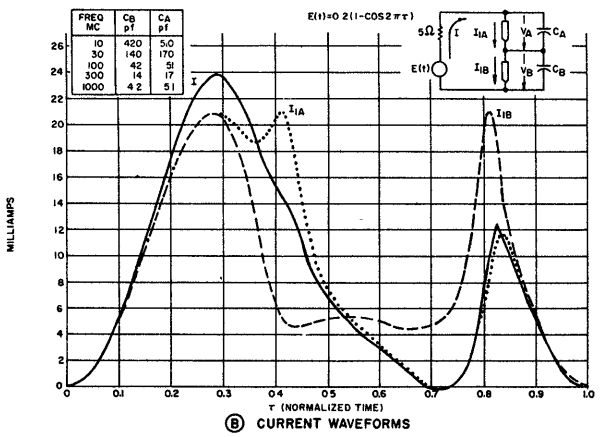
where: f_I is the frequency for which the waveforms were originally calculated, and C_{AII}, C_{BII}, and f_{II} are the new values of capacity and frequency for which the waveforms are also valid.

From the voltage waveforms it is seen that switching is quite apparent at 10 and 30 mc. (Figures 7A and 8A). Although some switching is evident at 100 mc. (Figure 9A), it is not distinct. However, at 300 mc. (Figure 10A) the diode capacity effectively shunts the nonlinear element and switching does not occur. Also note that at 300 mc. the diode voltages are almost sinusoidal. Examining the waveforms of Figure 6 which neglect capacitance effects, in conjunction with the waveforms of Figures 7 through 10, the effects of caacity become more clear. From the current waveforms it is seen that as the frequency increases, the capacity retards the appearance of sharp irregularities in the curves.

** Both computer solutions were obtained with the assistance of R. W. Klopfenstein, Director of Mathematical Services, and G. B. Herzog; RCA Laboratories, Princeton, N. J.

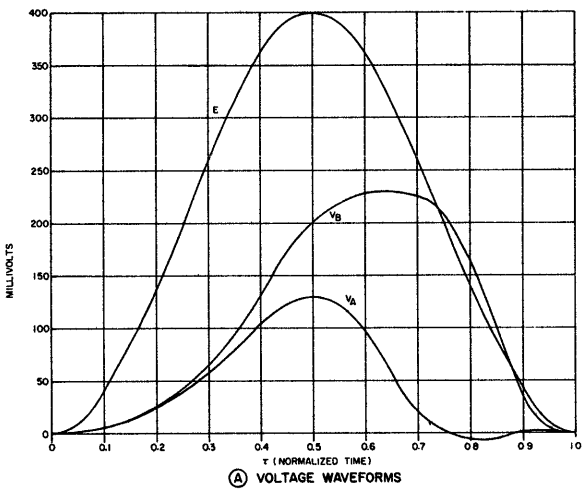


(A) VOLTAGE WAVEFORM

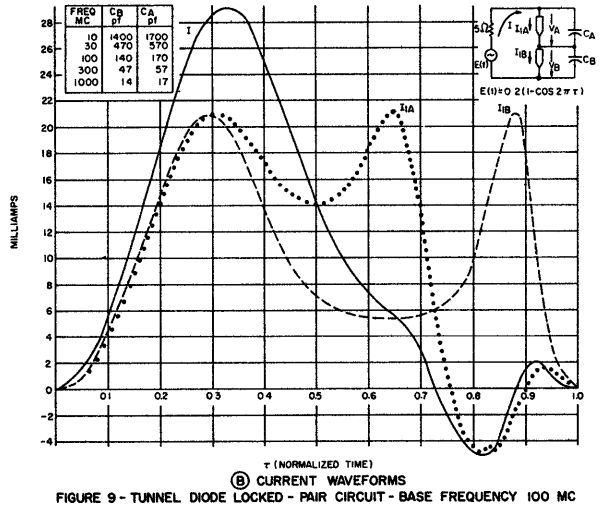


(B) CURRENT WAVEFORMS

FIGURE 8 - TUNNEL DIODE LOCKED-PAIR CIRCUIT
BASE FREQUENCY 30 MC



(A) VOLTAGE WAVEFORMS



(B) CURRENT WAVEFORMS

FIGURE 9 - TUNNEL DIODE LOCKED - PAIR CIRCUIT - BASE FREQUENCY 100 MC

C. Locked-Pair Circuit with Loading

From the previous discussion it should be noted that the two possible voltage outputs for the curves of Figure 3 are positive. To perform majority logic, it is desirable to drive the locked-pair circuit from both ends with voltages of equal magnitude but opposite polarity. The two possible voltage outputs of the circuit will then be equal in magnitude but opposite in sign.

A typical locked-pair majority gate is shown in Figure 11A. The locked-pair is controlled by the majority of n inputs (n must be odd), and in turn will deliver signals to m number of locked-pair circuits. The input-output impedance of the locked-pair circuit will be neglected as it is small in comparison with the coupling resistor. For majority rule, all but one input current may be cancelled. Hence, the sum of the current inputs to the locked pair at a minimum is V_O/R_C where: V_O represents the output voltage of a locked-pair circuit and R_C is the value of a coupling resistor. The equivalent circuit of the majority gate is shown in Figure 11B.

The circuit of Figure 11 was solved on the computer for a fan-in and fan-out of six. (m = n = 3). As in the first problem, equal tunnel diode V-I characteristics were assumed. The defining equations of the circuit in Figure 11b are:

$$E = K(1 - \cos 2\pi\tau) = I_A R_g + V_A + V_o$$

$$E = K(1 - \cos 2\pi\tau) = I_B R_g + V_B - V_o$$

$$V_o = I_o R_o = V_s - I_s R_s$$

$$V_s = \frac{K}{3}\left[1 - \cos 2\pi\left(\tau + \frac{1}{3}\right)\right]$$

$$I_A = f C_a \frac{dV_A}{d_\tau} + I_{1A};\ I_B = fC_B \frac{dV_B}{d_\tau} + I_{1B}$$

$$I_A = I_s = I_o + I_B$$

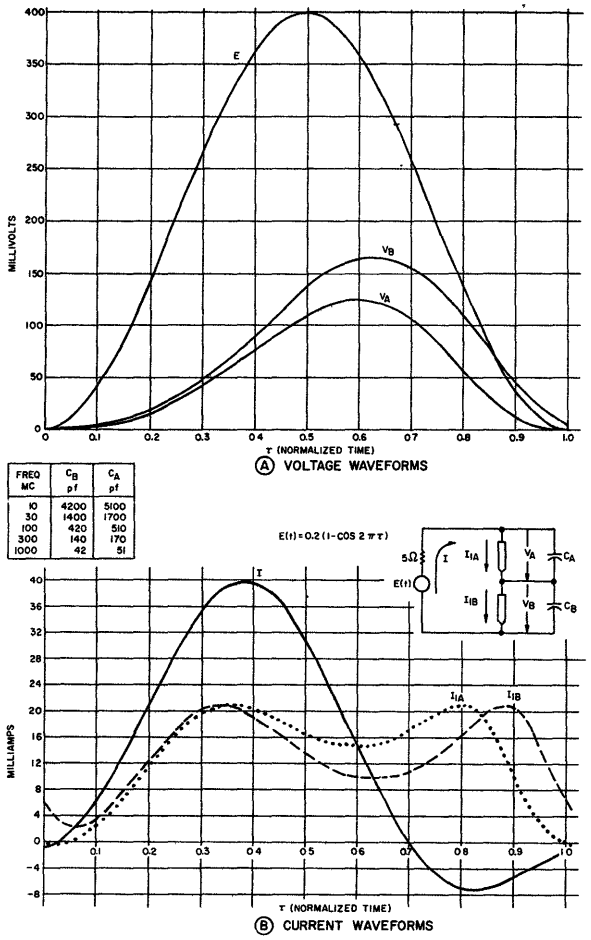
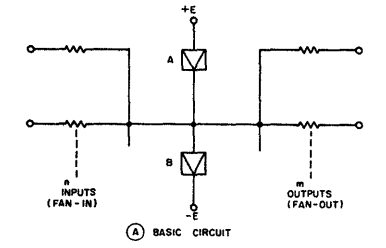


FIGURE 10 – TUNNEL DIODE LOCKED-PAIR CIRCUIT – BASE FREQUENCY 300 MC



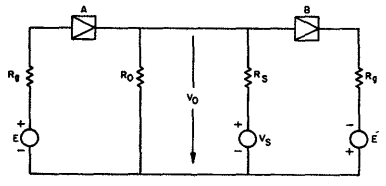


FIGURE 11 – LOCKED-PAIR MAJORITY GATE

where:

K = 0.10 volts

R_g = 5 ohms, the source resistance

$C_A = C_B$ = 20 $\mu\mu$f

$R_o = \frac{R_c}{m}$ = 50 ohms, the equivalent load or output resistance

$R_s = \frac{R_c}{n}$ = 50 ohms, the equivalent locking signal resistance

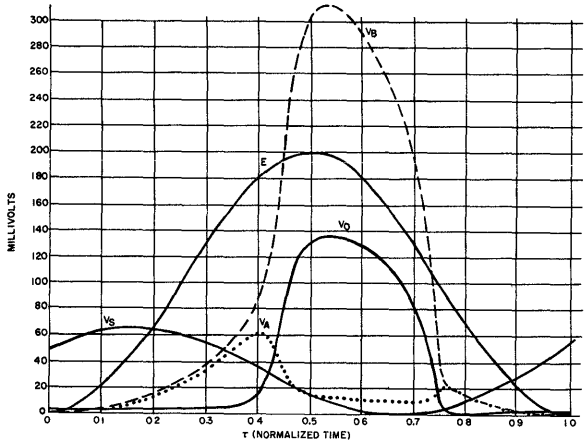
$V_s = \frac{E}{3}$, the simulated equivalent locking signal for n = 3

τ = ft, the normalized time

The smaller values of capacity used indicate that the diodes will switch at higher values of frequency than those used in the first problem; therefore, frequency values of 100, 300 and 600 mc. were used in this problem. As in the first problem, capacity and frequency appear only as a product in the circuit equations; thus, the waveforms become valid for other capacity and frequency values. The simulated locking signal, V_S, is optimistic since it assumes the maximum output voltage of the locked-pair circuit is equal to the maximum value of the source voltage E. Also, the locking signal is assumed to lead the source voltage by 120°; however, the actual locking signal is not only a function of the switching delay induced by the source resistance, R_g, but also is a function of the phase shift caused by the capacitors. This may be seen by examining the output voltage and current waveforms in Figure 12 through 14.

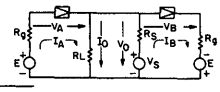
From the figures, note that the output voltage waveforms would barely be able to control the next stage because the next stage lags by $\tau/3$. For more positive control, the output voltage waveform should be either shifted to the right .1τ to .15τ or be maintained for a longer portion of the cycle. Larger capacity would yield some positive phase shift, but only at the sacrifice of amplitude. A decrease in source resistance would increase the effective period of the output voltage; but from practical considerations, 5 ohms is already small. One solution is to increase the source voltage. An attractive solution would be to use a larger source voltage and have it clipped.

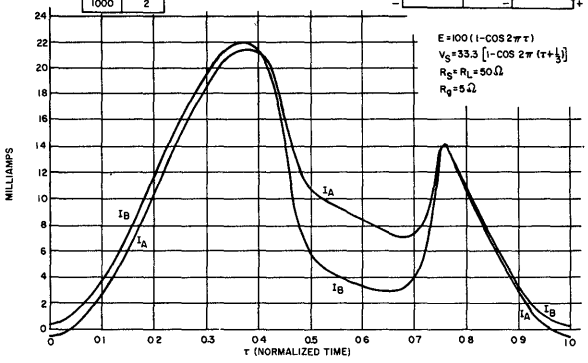
In general, the current and voltage waveforms for this problem are what would be expected after examining the waveforms of the first problem, i.e., the waveforms become "smoother" and the voltage waveform of the diode in the high state (diode B) decreases in amplitude as the frequency is increased. The output voltage at 600 mc. is almost zero. Although at 600 mc., the neglect of inductance may not be entirely valid, the waveforms indicate the restrictions placed on the circuit by diode capacity alone.



Ⓐ VOLTAGE

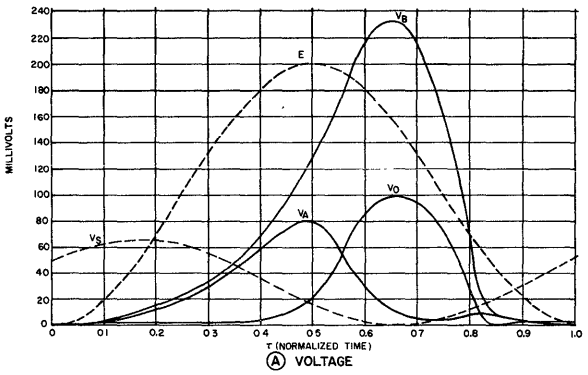
FREQ MC	$C_A=C_B$ pf
30	667
100	20
300	667
600	333
1000	2





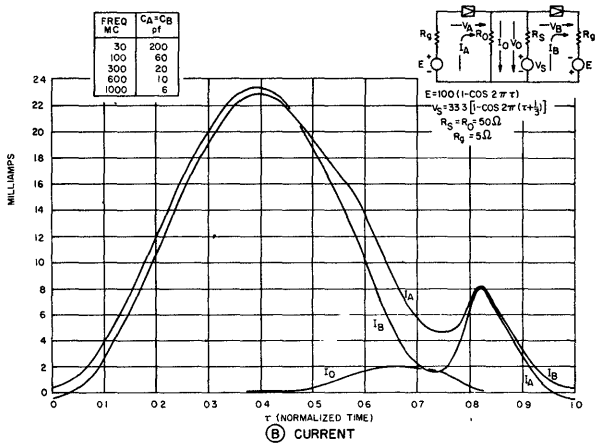
Ⓑ CURRENT

FIGURE 12-OUTPUT WAVEFORMS-BASE FREQUENCY 100 MC



Ⓐ VOLTAGE

FREQ MC	$C_A=C_B$ pf
30	200
100	60
300	20
600	10
1000	6



Ⓑ CURRENT

FIGURE 13-OUTPUT WAVEFORMS-BASE FREQUENCY 300 MC

The waveforms indicate that definite switching occurs for a diode capacitive reactance to maximum negative resistance ratio of 3 or greater. Consequently, a conservative relationship for germanium tunnel diodes in locked-pair circuits is:

$$\left[\begin{array}{c}\text{Maximum}\\ \text{switching frequency}\\ \text{in megacycles}\end{array}\right] = 250\left[\frac{\text{(peak current in milliamps)}}{\text{(capacity in } \mu\mu f)}\right]$$

Figure 15 shows the waveforms at 300 mc. for equal signal source and load resistors of 30 ohms. The only noticeable effect of this increased loading is to slightly decrease the output voltage. However, upon increasing the power supply resistance from 5 to 10 ohms, the diodes did not switch at all (Figure 16).

D. Graphical Interpretation of Loading

The effects of loading and source resistance, neglecting capacity effects, can be predicted by graphical means similar to those indicated in Part A of this paper. From the discussion of Figure 2, when element B went to its high voltage state, element A stayed in its low voltage state. Since the voltage across non-switching element A remains less than V_p (Figure 18), element A can be approximated by a resistance equal to V_p/I_p. Then, the equivalent circuit that is seen by active element B can be written from Figure 11. This Thevenin equivalent circuit is shown in Figure 17 where:

FREQ MC	$C_A=C_B$ pf
30	400
100	120
300	40
600	20
1000	12

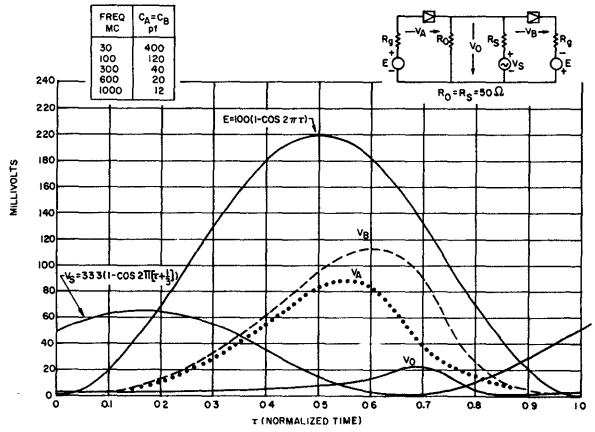


FIGURE 14-OUTPUT VOLTAGE WAVEFORMS-BASE FREQUENCY 600 MC

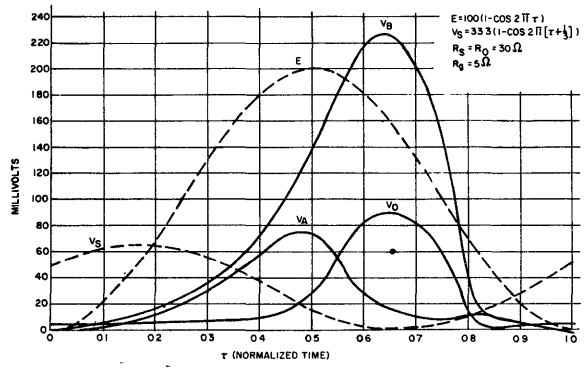


FIGURE 15-OUTPUT VOLTAGE WAVEFORMS WITH INCREASED LOADING-BASE FREQUENCY 300 MC

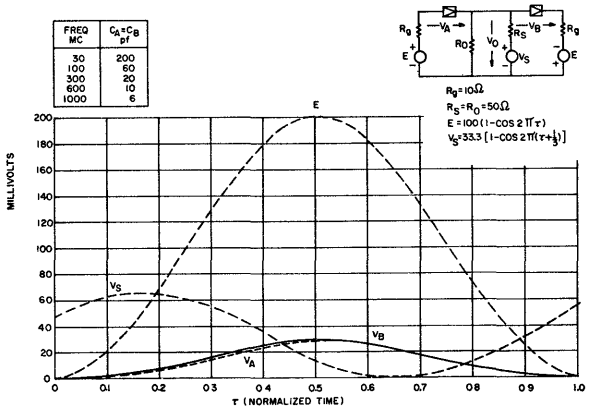


FIGURE 16-OUTPUT VOLTAGE WAVEFORMS WITH SOURCE RESISTANCE DOUBLED-BASE FREQUENCY 300 MC

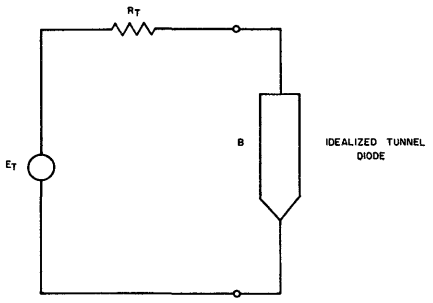


FIGURE 17-THEVENIN EQUIVALENT CIRCUIT OF FIGURE 11

$$E_T = E + \frac{ER_L}{R_g + R_L + r_A}, \text{ Thevenin equivalent voltage}$$

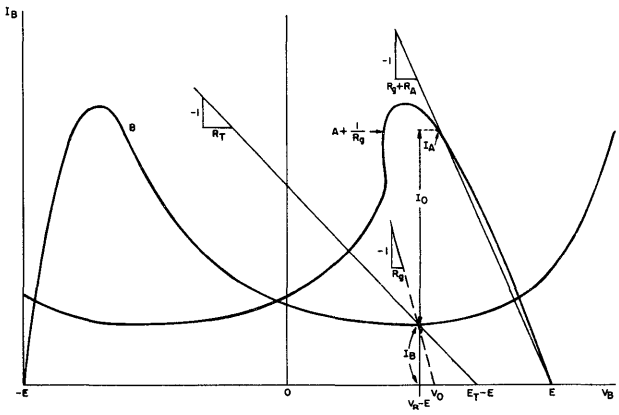


FIGURE 18-GRAPHICAL INTERPRETATION OF LOCKED-PAIR LOGIC GATE (FIGURE 11)

$$R_T = \frac{(R_g + r_A)\,R_L}{R_g + r_A + R_L} + R_g, \text{ Thevenin equivalent resistance}$$

$$r_A = \frac{V_{pA}}{I_{pA}}, \text{ linear approximation of element A for } V_A$$

$$< V_{pA}$$

$$R_L = \frac{R_s R_o}{R_s + R_o}, \text{ total loading}$$

The current-voltage relations in this equivalent circuit are shown in Figure 18. The intersection of the B characteristic with the load line, R_T, yields the voltage drop and current of the active element, B. Once these values are known, the rest of the circuit variables can be determined.

A graphical analysis was performed for the parameters used in the second computer solution. The results are plotted in Figure 19 for the source voltage, E, equal to its maximum value of 200 millivolts. These results predict quite accurately (to within 1% for the 100 mc. computer solution, Figure 12A) the maximum output voltage and the voltage across diode B. The maximum output voltage of the locked-pair circuit can be approximately determined by the relationship:

$$V_O = E_T - E - I_{VB}(R_T - R_g)$$

where: I_{VB} is the valley current of the tunnel diode that is in its high voltage state, and the other parameters are as previously defined.

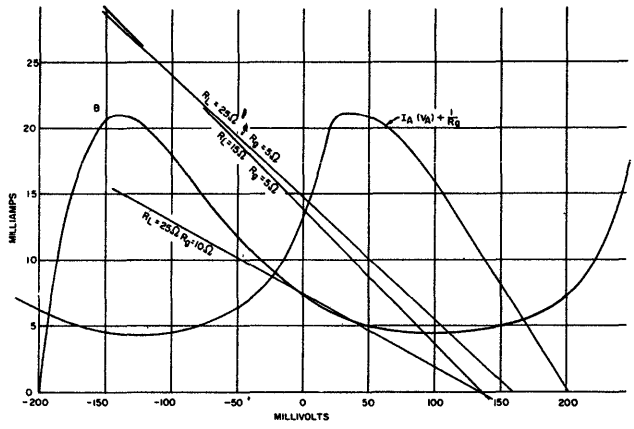


FIGURE 19- GRAPHICAL INTERPRETATION OF CIRCUIT IN FIGURE 11 USING ANALYTIC APPROXIMATION OF TUNNEL DIODE CHARACTERISTIC (SAME PARAMETERS AS USED FOR 2nd COMPUTER SOLUTION)

This equation was obtained from Figure 18, and was based on the assumption that the current in the active element is near the valley of the V-I characteristic. This approach may be used where the diode capacitive reactance is at least ten times the magnitude of the maximum negative resistance.

From Figure 19 it can be seen that the load line, R_T, changes very little when the total load resistance is decreased from 25 to 15 ohms, as was noted in the corresponding computer waveforms of Figures 13a and 15. Also predicted in Figure 19, is the fact that the diodes will not switch when the source impedance is increased from 5 to 10 ohms, as seen in the computer waveforms of Figure 16. The criteria for diode switching, as affected by loading and source resistance, is determined from Figure 18.

$$R_T < \frac{E_T - V_{pB}}{I_{pB}}$$

where: V_{pB} and I_{pB} are the coordinates at the point of peak current from the characteristic of the diode that is expected to go to its high state.

All the calculated results agree with the unpublished experimental data, and show that a digital computer is a powerful aid in the analysis of such complex nonlinear circuit problems.

Reference

1. Personal Communication with A.W. Lo.
2. "Esaki diode High-Speed Logical Circuits." E. Goto, IRE Transactions on Electronic Computers, Vol. EC-9, March 1960.
3. "Semiconductor Diodes in Parametric Subharmonic Oscillators." J. Hilibrand and W.R. Beam, RCA Review, June 1959.
4. "A New Concept in Computing." R.L. Wigington, IRE Proceedings, April 1960.
5. "Negative Resistance Elements as Digital Computer Component." Morton H. Lewin, Eastern Joint Computer Conference Proceedings 1959.

ON ITERATIVE FACTORIZATION IN NETWORK ANALYSIS BY DIGITAL COMPUTER*

W.H.Kim, D.H. Younger
Department of Electrical Engineering
Columbia University
New York 27,N.Y.

C.V. Freiman**, W. Mayeda***
International Business Machines Corporation
Yorktown Heights, New York

Abstract

The need to determine the sum of all tree admittance products occurs in almost all applications of topological network theory. This paper describes a method of obtaining this sum through an iterative factorization of the sum of tree admittance products of successively more complex subnetworks. Computational efficiency is achieved in that: (1) it is not necessary to test sets of branches for the presence of circuits; and (2) it is not necessary to calculate each tree admittance product.

A digital computer program has been developed for use on an IBM-704 which accommodates any network with complex branch admittances and up to 14 nodes. Far more complex networks may be analyzed, however, if they are first decomposed into two-terminal subnetworks. A detailed description and flow chart of the program are included.

Introduction

Although classical network theory has long permitted the analysis of arbitrarily large networks of prescribed form, it is only through the use of topological methods and high-speed digital computers that it has become possible to analyze a general RLC-network of any appreciable size. At a given level of theoretical and mechanical development, the practical limit on the size and complexity of a network which may be analyzed is largely determined by the efficiency of the algorithms used to implement the various topological network theorems. This paper seeks to improve this efficiency through use of a new method for computing the sum of tree admittance products -- a quantity used in almost all applications of topological network theory.

* This work was supported by National Science Foundation Grants G-3676 and G-6020.

** Formerly at Department of Electrical Engineering, Columbia University, New York, N.Y.

***Presently at Department of Electrical Engineering, University of Illinois, Urbana, Illinois.

Let us consider an RLC-network of e elements and n nodes, and let us use N to denote the graphical representation of the network. A number of authors[1,2,3] have discussed the generation of the sum of tree admittance products in N,T(N), and, in general, the methods they propose are based on:

(1) Determination of the trees of N by testing each combination of edges taken (n-1) at a time for the presence of circuits; and

(2) Calculation of the admittance product corresponding to each of the trees determined above.

The method we shall present utilizes the decomposition procedure introduced in [4] to compute T(N) in terms of the sums of tree admittance products of the various subgraphs of N. In practice, T is first determined for 3-node subgraphs, then for 4-node subgraphs. This iterative process continues until T(N) has been calculated. In this way, it is never necessary to test for the presence of circuits, and no admittance product is ever individually computed.

1. Iterative Determination of the Sum of Tree Products

Let N represent a connected, non-oriented weighted graph with n nodes and e edges. (The term "branch" may be substituted occasionally for "edge" but the term "element" will only be used in the sense of "element of a network" or "element of a set.") The sum of the weights of all edges which are incident on both node i and node j ($i \neq j$) will be denoted by w_{ij}. The method we shall use to determine T(N) is based on the following iterative procedure.

1. Select a generating node, say node n.

2. Partition the remaining (n-1) nodes into k subsets in all possible ways for k = 1, 2, ..., n-1. (Example A illustrates the partitioning for a particular 4-node network.)

Example A

Node 1 is chosen as the generating node. (See Fig. 1.)

k	Identification Number of Partition	Identification Number of Partition Element 1	2	3
1	1	(432)		
2	2	(32)	(4)	
	3	(43)	(2)	
	4	(42)	(3)	(2)
3	5	(4)	(3)	(2)

Table I. Partitioning of the Set of Nodes (432)

3. Associate subgraphs with each element of each partition as follows. An edge of N will be included in $N_{\mu\nu}$, the subgraph associated with the v-th element of the μ-th partition, if and only if both of the nodes upon which it is incident are contained in the v-th element of the μ-th partition. (It is sufficient that orderings of both partitions and elements exist. The nature of these orderings is unimportant.) Fig. 2 presents the various $N_{\mu\nu}$ which are derived from the graph of Example A.

4. We may now express the sum of tree products for graph N as

$$T(N) = \sum_{\mu} \left\{ \prod_{\nu} [T(N_{\mu\nu}) \sum_{i \in N_{\mu\nu}} w_{ni}] \right\} \quad (1)$$

where $T(N_{\mu\nu})$ is the sum of tree products for subgraph $N_{\mu\nu}$, n is the generating node, and i is any node in $N_{\mu\nu}$. If $N_{\mu\nu}$ consists of a single node (e.g. N_{32} of Example A), then $T(N_{\mu\nu}) = 1$; if $N_{\mu\nu}$ is non-connected, $T(N_{\mu\nu}) = 0$.

The maximum possible number of nodes in $N_{\mu\nu}$ is (n-1). Therefore, the problem of finding tree products in an n-node network has been changed into one of finding tree products in networks with at most (n-1) nodes. If (1) is applied to N_A, Fig. 1, as partitioned in Example A, we determine the sum of tree products of N_A to be

$$T(N_A) = T(N_{11})\cdot(e+f) + T(N_{21})\cdot(e+f)\cdot T(N_{22})\cdot 0 + T(N_{31})\cdot e\cdot T(N_{32})\cdot f + T(N_{41})\cdot f\cdot T(N_{42})\cdot e + T(N_{51})\cdot 0\cdot T(N_{52})\cdot e\cdot T(N_{53})\cdot f \quad (2)$$

$$= T(N_{11})\cdot(e+f) + T(N_{31})T(N_{32})\cdot ef + T(N_{41})T(N_{42})\cdot ef \quad (3)$$

At this point we observe that $T(N_{32}) = T(N_{42}) = 1$, $T(N_{31}) = d$, and $T(N_{41}) = a$. We may reuse (1) to obtain $T(N_{11})$ as

$$T(N_{11}) = (b+c)(a+d) + ad \quad (4)$$

Substituting (4) in (3) we have

$$T(N_A) = [(b+c)(a+d) + ad](e+f)+def+aef \quad (5)$$

$$= bae + bde + cae + cde + ade + baf + bdf + caf + cdf + adf + def + aef \quad (6)$$

If equation (1) is applied to a 3-node graph, the sum of tree products of any subgraph is either 1 or a sum of edge-weights (e.g. $T(N_{21}) = b+c$ in Fig. 2b). Thus, in actual computation, we first calculate the sum of tree products for all 3-node subgraphs of N which do not contain the generating node. These are then used to find the sum of tree products for 4-node subgraphs. This process is repeated until we have calculated T(N) itself. Example B demonstrates the actual method of computation, but, for clarity, does not completely reflect a computer solution. In Appendix B, the graph of Fig. 3 is solved in the manner of the computer solution.

Example B

(See Fig. 3). In any subgraph of N_B, the highest numbered node will arbitrarily be chosen to be the generating node. The symbol T(abc ... k) will denote the sum of tree products for the subgraph which is formed from the original graph by removing nodes other than abc ... k and all edges incident on these deleted nodes. Thus, in N_B, $T(124) = af$.

$$T(123) : T(12)\cdot(c+b+e) + (b+c)e = f(b+c+e) + (b+c)e \quad (7)$$

$$T(124) : fa$$

$$T(134) : ed$$

$$T(234) : (b+c)(a+d) + ad \text{ (Eq. 4)} \quad (8)$$

$$T(1234) : [(b+c)(a+d) + ad](e+f) + def + aef \text{ (Eq. 5)} \quad (9)$$

These constitute the sums of tree products for all subgraphs of N_B which do not contain node 5. We now determine $T(N_B)$ to be

$$T(N_B) = T(12345) = T(1234)\cdot(g+h) + T(124)\cdot hg + T(234)\cdot hg + T(12)h\cdot T(34)g + T(14)\cdot h\cdot T(32)g \quad (10)$$

$$T(N_B) = \{[(b+c)(a+d) + ad](e+f) + def + aef\}(g+h) + fahg + [(b+c)(a+d) + ad]hg + fhdg \quad (11)$$

As each edge of N_B has been uniquely identified, expanding (11) would result in an enumeration of all of the trees of N_B. On the other hand, if each edge were assigned a numerical value and only $T(N_B)$ were desired, only those equations corresponding to (1) would be employed. $T(N_B)$ is calculated in Example C for a particular assignment of numerical values to the edges of N_B.

Example C

In Fig. 3, let

$$a = c = e = g = 1$$

and

$$b = d = f = h = 2$$

$T(123) = 2(2+1+1) + (2+1)1 = 11$

$T(124) = 1 \cdot 2 = 2$

$T(134) = 1 \cdot 2 = 2$

$T(234) = (2+1)(1+2) + 1 \cdot 2 = 11$

$T(1234) = 11(1+2) + 2 \cdot 1 \cdot 2 + 1 \cdot 1 \cdot 2 = 39$

$T(N_B) = 39(1+2) + 2 \cdot 1 \cdot 2 + 11 \cdot 2 \cdot 1 + 2 \cdot 2 \cdot 2 \cdot 1 = 151$

Equation (11) results from the repeated application of (1) to N_B and various of its subgraphs. Some idea of the effort required to apply equation (1) to graphs of various sizes may be had by consulting Table II where the number of unique partitions for n-node graphs ($n = 1,2,\ldots, 13$) are tabulated. If the original graph contained seven nodes, then, after removing the generating node, all possible 3,4,5 and 6-node graphs would have to be inspected. From Table II, we see that a 6-node graph may be partitioned in 203 unique ways. The corresponding numbers for 2,3,4 and 5-node graphs are found to be 2,5,15 and 52. Thus, the number of different partitions which must be considered is found to be

$$\binom{6}{3}\cdot 2 + \binom{6}{4}\cdot 5 + \binom{6}{5}\cdot 15 + \binom{6}{6}\cdot 52 + 1\cdot 203 = 460 \quad (12)$$

When calculations involving this many partitions are to be carried out, it becomes essential that a reliable method of generating partitions be found. This could lead to the use of a table of partitions but, for computer calculations, the following procedure is preferable.

1. Choose one node, say node 1. It can be partitioned in only one way.

2. Add one node to the previously partitioned set of nodes increasing the number of nodes from k to k+1. For each partition of the k nodes into α subsets ($\alpha = 1,2,\ldots,k$)

	β: number of elements in set to be partitioned												
α: number of partition elements	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2		1	3	7	15	31	63	127	255	511	1023	2047	4095
3			1	6	25	90	301	966	3025	9330	28501	86526	265720
4				1	10	65	350	1701	7770	34105	145750	611501	2532530
5					1	15	140	1050	6951	42525	246730	1379400	7508501
6						1	21	266	2646	22827	179487	1323652	9321312
7							1	28	462	5880	63987	627396	5715424
8								1	36	750	11880	159027	1899612
9									1	45	1155	22275	359502
10										1	55	1705	39325
11											1	66	2431
12												1	78
13													1
TOTALS	1	2	5	15	52	203	877	4140	21147	115975	678570	4213597	27648528

Table II: The Number of Unique Ways in Which Sets of β Elements Can Be Partitioned Into α-element Partitions ($n_{\alpha\beta}$)

Note that $n_{\alpha,\beta} = \alpha \cdot n_{\alpha,\beta-1} + n_{\alpha-1,\beta-1}$ $\quad (\alpha,\beta = 1,2,3, \ldots)$

form α+1 partitions of the k+1 nodes as follows. Form one partition of (α+1) elements by using the new node as an additional element; form α partitions of α elements by adding the new node to each of the α elements of the original partition in turn.

3. Repeat 2 until all nodes have been partitioned.

If the sums of tree products for all newly developed subgraphs are generated at each step, then this method allows a straightforward determination of the sum of tree products for the overall graph. Example D illustrates this for N_B, the graph of Fig. 3.

Example D

Graph N_B of Fig. 3

nodes selected	partitions	sum of tree products of newly introduced subgraphs
1	(1)	
1,2	(1)(2)	
	(12)	
1,2,3	(1)(2)(3)	
	(13)(2)	
	(1)(23)	
	(12)(3)	
	(123)	T(123) = f(b+c+e)+(b+c)e
1,2,3,4	(1)(2)(3)(4)	
	(14)(2)(3)	
	(1)(24)(3)	
	(1)(2)(34)	
	(13)(2)(4)	
	(134)(2)	T(134) = ed
	(13)(24)	
	(1)(23)(4)	
	(14)(23)	
	(1)(234)	T(234) = (b+c)(a+d)+ad (Eq. 8)
	(12)(3)(4)	
	(124)(3)	T(124) = fa
	(12)(34)	
	(123)(4)	
	(1234)	T(1234) =
		= [(b+c)(a+d)+ad](e+f) + def + aef

$T(N_B)$ may now be calculated as in Eq. (11).

2. Computer Determination of the Sum of Tree Admittance Products

The digital computer program discussed in this section was written for use on an IBM 704 and is designed to calculate the sum of tree admittance products for an electrical network in which the branch admittances are expressed as complex numbers. For input purposes these network admittances must be combined in such a form that no more than one branch exists between any pair of nodes. The program will accept any network with as many as fourteen nodes provided a 32,768-word magnetic core storage unit is available.

A block diagram showing the functional steps in the overall calculation is given in Flow Chart I. Appendix A supplies details about the format and execution of the various functions. Appendix B demonstrates the method by which the program handles the calculation of T(N) for the network considered in Section 1, Example B.

Note that the program makes a distinction between those nodes connected to the generating node and those which are not. After a subgraph has been selected and a generating node chosen, the initial partitioning does not involve all the remaining nodes of the subgraph but only those nodes which are connected to the generating node. The unconnected nodes are then distributed among the elements of each partition in all possible ways. The possibility of ever having

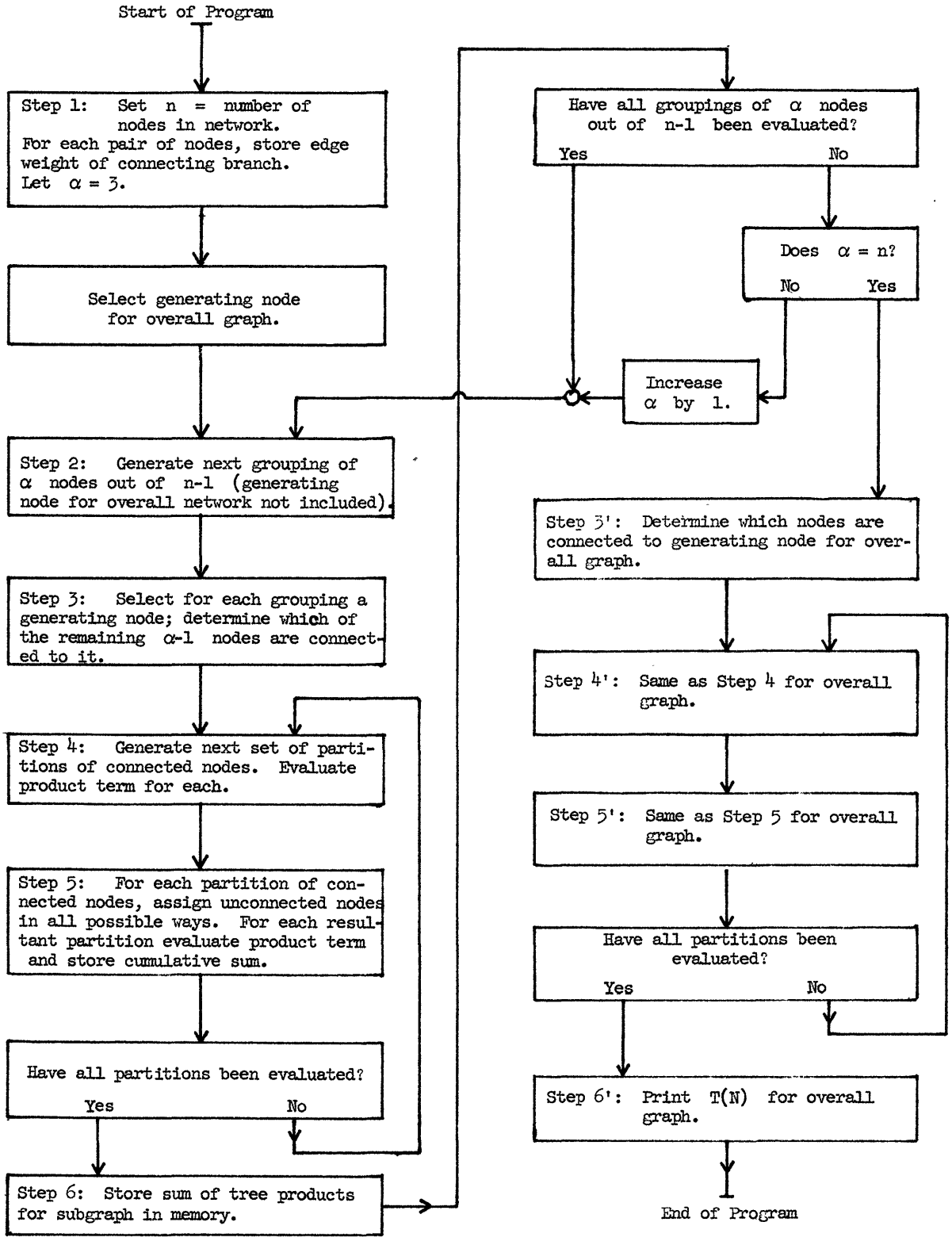
$$\sum_{i \in N_{\mu v}} w_{ni} = 0 \qquad (13)$$

is avoided in this way.

The two potential limitations on any program which calculates the sum of tree admittance products are memory capacity and computer time. Either or both will limit the size and generality of the network which may be analyzed.

The problem of memory capacity is revealed by the table of partitions, Table II. For $n \geq 9$, there are simply too many partitions for them all to be developed in the computer core memory at one time. The need to conserve computer time prohibits the use of any of the other computer storage devices. To perform such an extensive set of calculations with a limited storage capacity, the program must use an efficient procedure which generates a few partitions, evaluates these and stores the partial sum, generates a few more, etc., until all partitions have been generated and evaluated. A re-examination of the method used in generating the partitions of Example D indicates such a procedure. In that example, the partition involving node 1 alone is developed, from this partition all partitions involving nodes 1 and 2 are developed, all the partitions involving nodes 1, 2, and 3 are developed from these, and finally all partitions involving four nodes are developed. In general, the method generates all partitions involving k+1 nodes from a knowledge of all those involving k-nodes. However, it is possible at any stage to find those k+1 node partitions corresponding to only one of the k-node partitions, then to find those k+2 node partitions corresponding to only one of the k+1 node partitions, etc. When all the partitions generated by a single k-node partition have been utilized, then the k+1 node partitions corresponding to another k-node partition are gene-

Flow Chart I


Start of Program
Step 1: Set n = number of nodes in network. For each pair of nodes, store edge weight of connecting branch. Let α = 3.
Select generating node for overall graph.
Step 2: Generate next grouping of α nodes out of n-1 (generating node for overall network not included).
Step 3: Select for each grouping a generating node; determine which of the remaining α-1 nodes are connected to it.
Step 4: Generate next set of partitions of connected nodes. Evaluate product term for each.
Step 5: For each partition of connected nodes, assign unconnected nodes in all possible ways. For each resultant partition evaluate product term and store cumulative sum.
Have all partitions been evaluated?
Yes
No
Step 6: Store sum of tree products for subgraph in memory.
Have all groupings of α nodes out of n-1 been evaluated?
Yes
No
Does α = n?
No
Yes
Increase α by 1.
Step 3': Determine which nodes are connected to generating node for overall graph.
Step 4': Same as Step 4 for overall graph.
Step 5': Same as Step 5 for overall graph.
Have all partitions been evaluated?
Yes
No
Step 6': Print T(N) for overall graph.
End of Program

rated, and these made use of in turn. In this manner, all the partitions may be generated.

Perhaps the method will be clarified by reworking example D. The object is to develop the fifteen 4-node partitions in such a way as to retain the smallest number of partitions in memory at any one time. In Example E, which shows the successive stages of the development of these partitions, X indicates that the product term is evaluated at that stage and added to the cumulative sum. Note that at no stage is it necessary to retain more than five partitions in memory.

Example E

Development of 4-node partitions by an iterative procedure, illustrating a saving in memory capacity.

$$(1) \rightarrow \begin{matrix}(1)(2)\\(12)\end{matrix} \rightarrow \begin{matrix}(1)(2)\\(12)(3)\\(123)\end{matrix} \rightarrow \begin{matrix}(1)(2) & \\ (12)(3) & \rightarrow \\ (123)(4) & X \\ (1234) & X\end{matrix}$$

$$\rightarrow \begin{matrix}(1)(2) & \rightarrow & (1)(2)(3) & & (1)(2)(3) & \\ (12)(3)(4) & X & (13)(2) & & (13)(2) & \rightarrow \\ (124)(3) & X & (1)(23) & \rightarrow & (1)(23)(4) & X \\ (12)(34) & X & & & (14)(23) & X \\ & & & & (1)(234) & X\end{matrix}$$

$$\rightarrow \begin{matrix}(1)(2)(3) & \rightarrow & (1)(2)(3)(4) & X \\ (13)(2)(4) & X & (14)(2)(3) & X \\ (134)(2) & X & (1)(24)(3) & X \\ (13)(24) & X & (1)(2)(34) & X\end{matrix}$$

The significance of this saving of core storage is better appreciated by considering a 12-node subgraph. Table II shows that there are over 4 million 12-node partitions. Yet it is possible to generate all of them and evaluate the product term for each without ever retaining as many as one hundred partitions at any time. These partitions, being so few in number, may be stored in memory in a convenient and efficient manner. For the sake of programming convenience, twelve storage locations are set aside to retain a single partition of 12 nodes; nonetheless, the total storage for partitions is so small as to be negligible.

In evaluating the product term for the partitions, a saving in computer time has been realized by organizing the computations to take advantage of the iterative development of the partitions. Consider once again the partition development given in Example E. The partition product for each 4-node partition may be evaluated separately, and the sum cumulatively stored. But each 4-node partition is derived from some 3-node partition by altering at most one partition element. If the 3-node partition product has been evaluated, each 4-node product derived from it may be evaluated by dividing out the old factor and multiplying by the factor which replaces it. For those partitions which are generated by adding the new node as a separate element, it is not even necessary to divide out any factor. The case in which one or more of the factors is zero must be treated carefully but provides no serious problems. The use of this procedure is illustrated in Appendix B. However, its usefulness is only appreciated by considering a partition of many elements -- say twelve. In evaluating such a partition we may, instead of performing the twelve multiplications of the element factors, perform one division and one multiplication. Of course, it is not just as simple as the above comparison would indicate, since we must provide additional bookkeeping and also evaluate the partition products for those intermediate partitions used in the generation.

Conclusions

The importance, in calculating the sum of tree products for any graph, of using a factored expression involving the sums of tree products of the subgraphs, can be seen in the following comparison. There are associated with a 12-node graph (or network) N in which an edge connects each pair of nodes 12^{10} or approximately 6.19×10^{10} trees. In analyzing this network by our method, the product term of Eq. (1), Section 1, must be evaluated for each of 1.52×10^{6} groupings of subgraphs. Thus in a 12-node network, there are about 4×10^{4} times as many trees as groupings. Even these groupings are not evaluated separately; the iterative nature of the development of the groupings permits evaluation of common factors of these products once for many groupings. Since there are for a given network many less groupings than individual trees and since even these groupings are evaluated in factored form, a program based on our method is potentially orders of magnitude faster than any method which requires the calculation of individual tree admittance products.

The method by which T(N) is calculated does not depend on any special structural characteristics (i.e. ladder or lattice structures, etc.) of the network N. Networks of completely arbitrary topology and branch weights may be analyzed provided the number of nodes in the network does not exceed the prescribed limit. However, networks of interest often contain subnetworks which are connected to the rest of the network through two terminals only. In such a case, the calculation can be shortened by determining the driving point admittance of the two terminal subnetwork, which then is replaced in N by a single branch whose weight is the complex admittance of the subnetwork at the frequency under consideration. The use of such simplification, where possible, will greatly reduce the computation time.

MacWilliams[2] cited an example of a network of 11 nodes and 21 branches which requires 30 minutes of IBM 704 time. Whether the network had any topological characteristics which simplified the computation

is not known; further, it is presumed that the edge weights of the branches were restricted to pure real or pure imaginary values. For a network with 11 nodes, and arbitrary topology, which may contain as many as 55 branches each of which has a complex weight, our program estimates less than 30 minutes in computation of the sum of tree admittance products.

There are several ways in which the efficiency and usefulness of the program may be extended. The first modification, a simple one, effects a considerable saving of computer time when the network under consideration reflects a minor change from the network previously investigated. After a particular grouping of α nodes out of n-1 has been generated, step 2 of Fig. 5, this subgraph is tested to see if any of its edges have been modified from those of the network previously analyzed. If none have been modified, the value for that subgraph will already be in memory and the program can proceed immediately to the next grouping.

A more general and useful calculation than that performed by the present program is to find the sum of tree admittance products as a function of frequency. That is, instead of the input edge weights expressed in terms of complex numbers, let these weights be given as a rational function of frequency. The resultant sum of tree products would then be expressed as a complex function of frequency. To solve this more general problem, certain modifications of the program are needed, particularly the method of evaluation of the partitions. None of the difficulties involved appear to be of a fundamental nature, although the complexity of the network which could be analyzed would be restricted.

A further extension is a program which considers any one of the network edges a variable admittance. With this additional modification the program could be used to directly calculate the input admittance of a given network looking in at any pair of terminals as a function of frequency.

Appendix A

Programming details and format are given for each of the functional steps indicated in Flow Chart I, Section 2.

Step 1

The value of n and the edge weights are supplied on punched cards. For the latter there is, for each pair of nodes in the network, a card on which is punched two numbers designating the nodes (such as 2-6) and also the real and the imaginary parts of the complex admittance of the branch connecting the nodes. Where no branch exists in the network between two given nodes, the value zero real and zero imaginary is supplied. For non-zero admittances, the values are given in floating point decimal using the standard decimal card code. The allowable range of floating point decimal numbers is approximately $\pm 1 \times 10^{-37}$ to $\pm 1 \times 10^{+37}$ and zero. With regard to these limits, some caution must be exercised since these limits apply not only to the input edge weights but to the numbers arising from all calculations in the program.

After translating these branch weights into floating point binary, the program stores them as part of a larger table in core memory. This table will eventually contain the sum of tree admittance products for all subnetworks; the input edge weights constitute the sum of tree products for their two-node networks.* This table will occupy two blocks of 2^{n-1} memory locations, one block for the real parts of the admittances, one for the imaginary parts. Note that the exponent is n-1 rather than n since only those subgraphs not including the generating node for the overall graph will be analyzed. It is the size of this table, 2^n locations, which limits the size of the network which may be analyzed. For $n = 14$, $2^n = 16,384$ locations or one-half of the core storage of a 32,768-word memory unit. This is one reason that the program is limited to networks of not more than fourteen nodes. The generating node will arbitrarily be chosen as the highest-numbered node.

Step 2

This step generates the various groupings of α nodes out of n-1. The format used throughout the program to indicate a given grouping of nodes is a simple one: the last n binary positions of the 36-bit computer word gives, in positional notation, the nodes considered. The subgraph, of a 6-node graph consisting of nodes 1, 3, 5 and 6 would be coded as shown in Fig. 4.

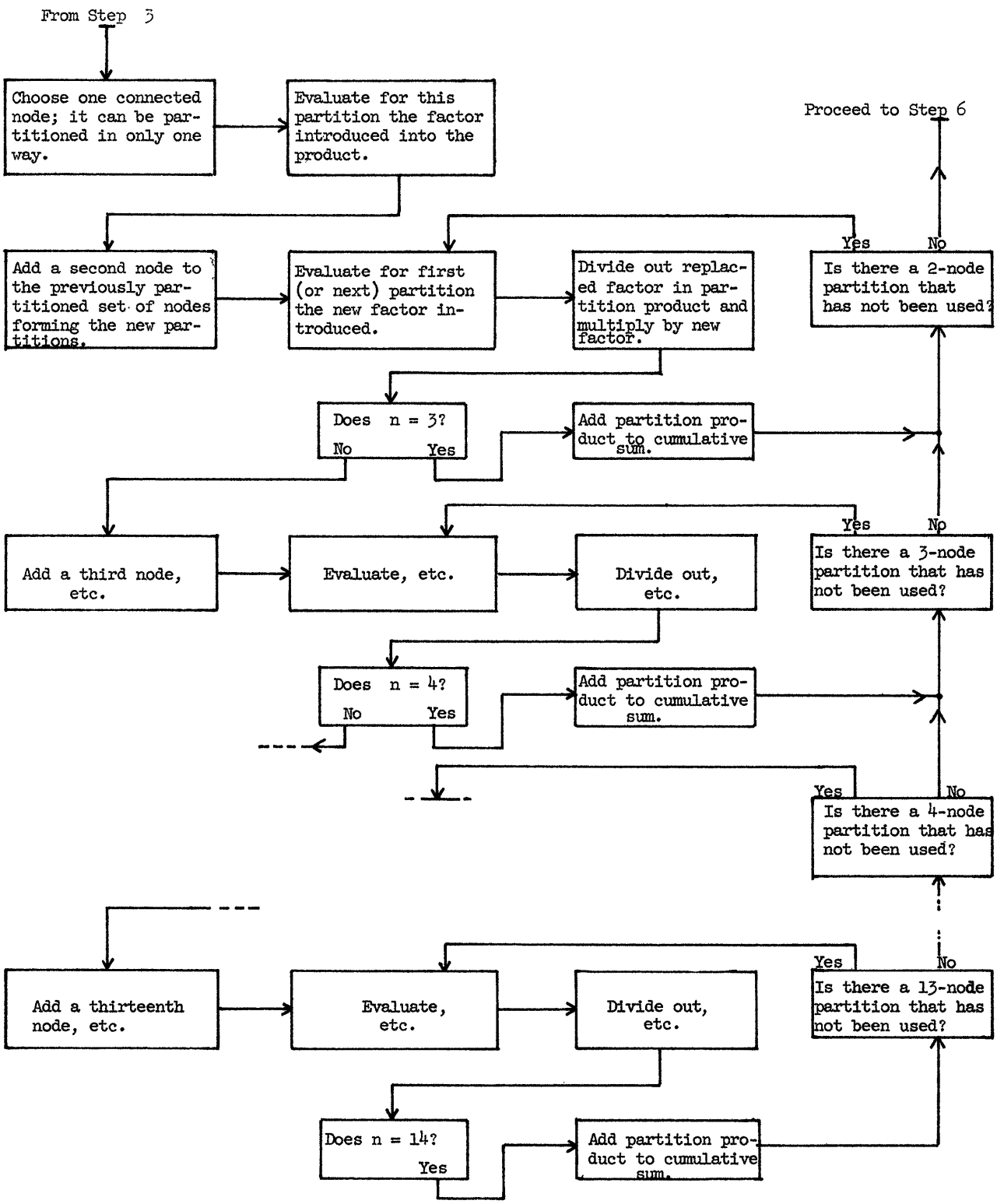
To generate all groupings of α nodes out of n-1, all binary numbers from 1 to 2^{n-1} are first generated, along with the binary weight of the number in the unused upper portion of the word. From this set of 2^{n-1} binary numbers are chosen, in turn, those of weight α (where $\alpha = 3,4,\ldots,n-1$). This is equivalent, according to the format, to choosing all groupings of α nodes out of n-1 for $\alpha = 3$, $\alpha = 4$, etc.

Step 3

It is necessary to select for each subgraph a generating node. The left-most binary unit corresponding to the highest numbered node is always selected. The remaining nodes are separated into two classes, those connected to the generating node, and those not connected. A node is considered connected if either the real or the imaginary part of the admittance of the branch connecting it to the generating node

* Actually the weights of input edges, where one node is the generating node for the overall graph, will have to be stored in a separate table.

Flow Chart II



is non-zero; if both real and imaginary parts are zero the node is considered unconnected.

Steps 4 and 5

The method of generating partitions for the connected nodes, with the distribution of the remaining nodes among the elements of each partition in all possible ways, and the evaluation of the product term for each, constitutes the core of the program. The structure of this section of the program is quite complex. The manner in which it faces the two potential limitations of the program, memory capacity and computer time, has been given in the body of the report.

To clarify the structure, a block diagram of these two steps is shown in Flow Chart II. In this diagram there is, for reasons of simplicity, no distinction shown between the treatment of the connected nodes and the unconnected nodes. The only distinction which the program makes between the two is that unconnected nodes are never permitted to occupy a partition element by themselves, since such a partition would make no contribution to the sum of tree products.

Step 6

After all the partitions have been generated, and evaluated, the cumulative sum, which is now the sum of tree products for the subgraph under consideration, is stored in the appropriate location of the table mentioned in connection with step 1. The program then proceeds to step 2 unless all groupings have been considered. In step 6', the sum of tree products for the overall graph is evaluated. It is translated back to floating point decimal and the result printed out. If the values of the sum of tree products for any subgraphs are desired, these may also be translated and printed out using the same routine.

Appendix B

The following analysis of the network of Fig. 3 closely reflects the actual computer solution. The numerical values of the edge are those used in Example C, Section 1.

$$a = c = e = g = 1$$

$$b = d = f = h = 2$$

The steps which are designated correspond to those given in the flow chart shown in Section 2, Flow Chart I.

Step 1

Set $n = 5$.

Input:

Nodes		Edge Weight Real	Imaginary
1	2	2	0
1	3	1	0
1	4	0	0
1	5	2	0
2	3	3	0
2	4	1	0
2	5	0	0
3	4	2	0
3	5	1	0
4	5	0	0

Select node 5 as generating node.

Step 2

Generate all binary numbers from 1 to 2^4 to determine all possible groupings of nodes 1, 2, 3 and 4.

Nodes 4	3	2	1	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	→ T(123)
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	→ T(124)
1	1	0	0	
1	1	0	1	→ T(134)
1	1	1	0	→ T(234)
1	1	1	1	→ T(1234)

Let $\alpha = 3$.

T(123)

Step 3 Choose node 3 as generating node; nodes 1,2 are connected.

Step 4

Partition	New factor in product term	Product term	Partial sum of products
(1)	$T(1)w_{31} = 1 \cdot 1 = 1$	1	-
(1)(2)	$T(2)w_{32} = 1 \cdot 3 = 3$	$1 \cdot 3 = 3$	3
(12)	$T(12)(w_{31}+w_{32}) = 2(1+3) = 8$	$\frac{1}{1} \cdot 8 = 8$	11

Step 6 Store T(123) = 11.

T(124)

Step 3 Choose node 4 as generating node; node 2 is connected, node 1 not connected.

Steps 4 and 5

	Partition	New factor in product term	Product term	Partial sum of products
4	(2)	$T(2)w_{42} = 1 \cdot 1 = 1$	1	-
5	(12)	$T(12)(w_{41}+w_{42}) = 2 \cdot (0+1) = 2$	$\frac{1}{1} \cdot 2 = 2$	2

Step 6 Store T(124) = 2.

T(134)

Step 3 Choose node 4 as generating node; node 3 is connected, node 1 not connected.

Steps 4 and 5

	Partition	New factor in product term	Product term	Partial sum of products
4	(3)	$T(3)w_{43} = 1 \cdot 2 = 2$	2	-
5	(13)	$T(13)(w_{41}+w_{43}) = 1(0+2) = 2$	$\frac{2}{2} \cdot 2 = 2$	2

Step 6 Store T(134) = 11.

T(234)

Step 3 Choose node 4 as generating node; nodes 2 and 3 are connected.

Step 4

Partition	New factor in product term	Product term	Partial sum of products
(2)	$T(2)w_{42} = 1 \cdot 1 = 1$	1	-
(2)(3)	$T(3)w_{43} = 1 \cdot 2 = 2$	$1 \cdot 2 = 2$	2
(23)	$T(23)(w_{42}+w_{43}) = 3(1+2) = 9$	$\frac{1}{1} \cdot 9 = 9$	11

Step 6 Store T(234) = 11

Let $\alpha = 4$.

T(1234)

Step 3 Choose node 4 as generating node; nodes 2, 3 are connected, node 1 not connected.

Steps 4 and 5

	Partition	New factor in product term	Product term	Partial sum of products
4	(2)	$T(2)w_{42} = 1 \cdot 1 = 1$	1	-
	(2)(3)	$T(3)w_{42} = 1 \cdot 2 = 2$	$1 \cdot 2 = 2$	-
	(23)	$T(23)(w_{42}+w_{43}) = 3(1+2) = 9$	$\frac{1}{1} \cdot 9 = 9$	-
5	(12)(3)	$T(12)(w_{41}+w_{42}) = 2(0+1) = 2$	$\frac{2}{1} \cdot 2 = 4$	4
	(2)(13)	$T(13)(w_{41}+w_{43}) = 1(0+2) = 2$	$\frac{2}{2} \cdot 2 = 2$	6
	(123)	$T(123)(w_{41}+w_{42}+w_{43}) = 11(0+1+2) = 23$	$\frac{9}{9} \cdot 33 = 33$	39

Step 6 Store $T(1234) = 39$.

$T(N_B) = T(12345)$

Node 5 is generating node.

Step 3' Nodes 1 and 3 are connected, nodes 2 and 4 not connected.

Steps 4' and 5'

	Partition	New factor in product term	Product term	Partial sum of products
4'	(1)	$T(1)w_{51} = 1 \cdot 2 = 2$	2	-
	(1)(3)	$T(3)w_{53} = 1 \cdot 1 = 1$	$2 \cdot 1 = 2$	-
	(13)	$T(13)(w_{51}+w_{53}) = 1(2+1) = 3$	$\frac{2}{2} \cdot 3 = 3$	-
5'	(12)(3)	$T(12)(w_{51}+w_{52}) = 2(2+0) = 4$	$\frac{2}{2} \cdot 4 = 4$	-
	(1)(23)	$T(23)(w_{52}+w_{53}) = 3(0+1) = 3$	$\frac{2}{1} \cdot 3 = 6$	-
	(124)(3)	$T(124)(w_{51}+w_{52}+w_{54}) = 2(2+0+0) = 4$	$\frac{4}{4} \cdot 4 = 4$	4
	(12)(34)	$T(34)(w_{53}+w_{54}) = 2(1+0) = 2$	$\frac{4}{1} \cdot 2 = 8$	12
	(14)(23)	$T(14)(w_{51}+w_{54}) = 0(2+0) = 0$	$\frac{6}{2} \cdot 0 = 0 \cdot 3$	12
	(1)(234)	$T(234)(w_{52}+w_{53}+w_{54}) = 11(0+1+0) = 11$	$\frac{6}{3} \cdot 11 = 22$	44
	(123)	$T(123)(w_{51}+w_{52}+w_{53}) = 11(2+0+1) = 33$	$\frac{3}{3} \cdot 33 = 33$	-
	(1234)	$T(1234)(w_{51}+w_{52}+w_{53}+w_{54}) = 39(2+0+1+0) = 117$	$\frac{33}{33} \cdot 117 = 117$	151

Step 6' Print $T(N_B) = 151$ Real
0 Imaginary

End of Program.

Bibliography

1. W. Mayeda and M.E. Van Valkenburg, "Network Analysis and Synthesis by Digital Computers," 1957 WESCON Convention Record, Pt. 2, pp. 137-144.

2. F.J. MacWilliams, "Topological Network Analysis as a Computer Program," IRE Trans. on Circuit Theory, vol. CT-5, pp. 228-229; September 1958.

3. E.W. Hobbs, "Topological Network Analysis as a Computer Program (discussion)", IRE Trans. on Circuit Theory, vol. CT-6, pp. 135-136; March, 1959.

4. W. Mayeda, "Reducing Computation Time in the Analysis of networks by Digital Computer," IRE Trans. on Circuit Theory, vol. CT-6, pp. 136-137; March 1959.

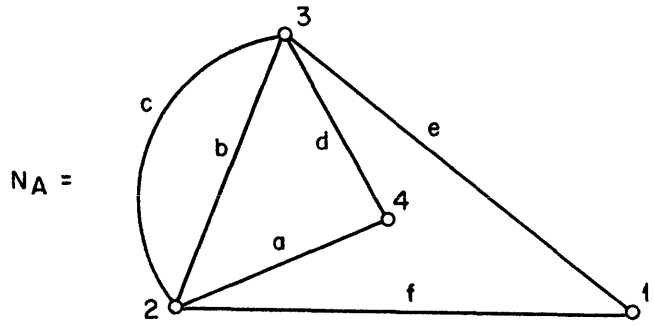
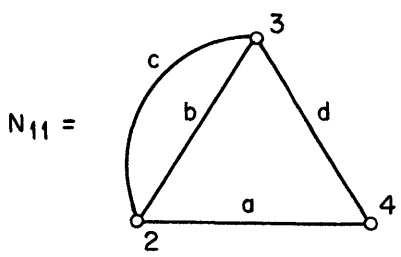
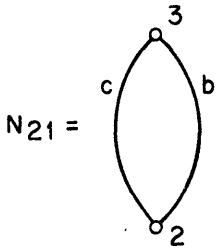


Fig. 1



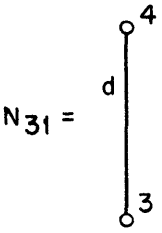
(a)



(b)

$N_{22} = o4$

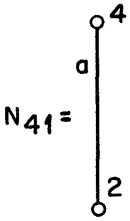
(c)



(d)

$N_{32} = o2$

(e)



(f)

$N_{42} = o3$

(g)

$N_{51} = o4$

(h)

$N_{52} = o3$

(i)

$N_{53} = o2$

(j)

Fig. 2

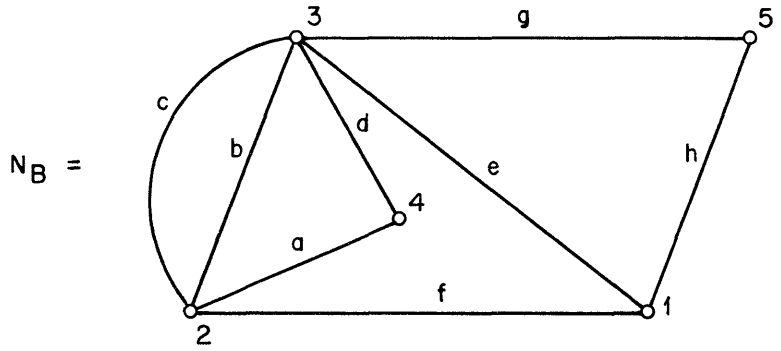
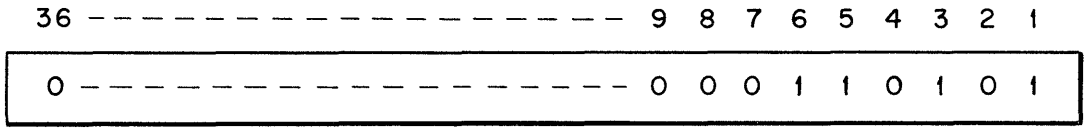


Fig. 3



CODING OF SUBGRAPH CONSISTING
OF NODES 1, 3, 5, AND 6

Fig. 4

A COMPUTER CONTROLLED DYNAMIC SERVO TEST SYSTEM

V. A. Kaiser and J. L. Whittaker
Douglas Aircraft Company, Inc.
Santa Monica, California

Recent years have seen an increasing number of successful applications of digital computers to a real-time control in industry. These applications generally concern the monitoring and control of a large number of relatively slowly varying parameters of a process. Equally successful has been an application of a control computer to provide control and data processing for the dynamic testing of missile servo-control systems. At Douglas Aircraft Company, computer-controlled tests are currently being conducted on servo systems which operate at frequencies ranging to 400 cps. In this application, emphasis is placed on the monitoring of several rapidly changing parameters in order to provide a real-time description of the dynamic characteristics of the servo systems.

PURPOSE OF THE AUTOMATIC TEST SYSTEM

Steadily increasing requirements are being placed on the performance, reliability, and operating environment of missile flight control servo systems. To satisfy these advanced requirements has demanded a continuous effort in the missile industry to improve on previous designs, maintain closer design parameters, and to develop better hardware. Each stage in the development of a missile flight control system requires extensive testing, and the development program cannot proceed until it has been ascertained that all intermediate design requirements have been met. These efforts are resulting in an ever-increasing amount of laboratory dynamic testing.

Manual methods of conducting the many required dynamic control system tests have become undesirable for several reasons. To complete a development program in the alloted time requires that several tests be conducted concurrently in order to offset the time required to conduct each individual test manually. This parallel testing procedure necessitates extensive duplication of test equipment and a large labor force of test engineers, technicians, and data reduction personnel. The manual methods of conducting the tests have thus become an economic burden. In addition, the reliability of the manual test results is relatively low. Errors accumulate from numerous sources such as faulty equipment calibrations, incorrect interpretations of low signal-to-noise ratio signals, and carelessness. Small changes in servo parameters due to wear, hydraulic fluid contamination, or operating environment may therefore be completely hidden in the "data spread".

These deficiencies in the manual methods of control system testing were recognized by the Testing Division of Douglas Aircraft Company. In early 1958, a study was initiated to investigate possible methods of improving the testing procedures. It was conceived that the duplication of test equipment and personnel required by the parallel testing procedure could be eliminated by time-sharing a single master test system. Economic justification of the master tester could be realized if the individual tests could be conducted serially in the same length of time as that required for parallel testing by the conventional methods. The study was therefore directed toward an automatic computer-controlled test system. The versatility necessary for development test work, as well as the speed, accuracy, and reliability requirements, limited the study to digital rather than analog equipment. From this study resulted a digital computer-controlled dynamic servo test facility which was placed in operation in mid-1960.

This automatic test system, controlled by a Thompson-Ramo-Wooldridge RW-300 digital control computer, has been located in the Hydro-Mechanical Systems Development Laboratory at Douglas. With this facility, the many various servo system tests which previously required weeks to perform are now conducted daily.

GENERAL FACILITY DESCRIPTION

The RW-300 computer and its associated input-output equipment are located in what is called the Control Center of the Systems Laboratory. From this central location, tests are conducted on missile servos located at eleven remote sites in the laboratory as shown in Figure 1. These sites include missile mock-ups, environmental chambers, and work benches. Switching is accomplished manually in the Control Center to select one of these sites, and the equipment located at the selected site is tested automatically by the computer. Intercom and closed circuit television systems provide means for communications between the operator in the Control Center and personnel at the test locations. The operator can thus observe the system in operation while the test is being performed and can immediately stop the test should a malfunction occur. With proper scheduling of tests at the various remote locations, many different tests on various servo-system hardware configurations can be conducted in succession.

The RW-300 computer is the heart of the automatic test system. Designed for process control, the computer contains a flexible input-output subsystem. Using the digital and analog outputs, control of the servo system configurations, the driving functions applied, and other peripheral equipment is maintained throughout the test (Figure 2). The analog-to-digital converter, an integral part of the RW-300 computer, is used to

convert samples of the servo system output signals to digital form. Conversion is under program control and the digital values of the samples are stored automatically in a reserved section of the magnetic drum storage of the computer.

The memory capacity of the computer is approximately 8000 words, including 1024 words reserved for analog data and 16 words of rapid-access memory. Each word contains 18 binary digits, and may be used for program storage (1 + 1 instruction system) or data storage (17 magnitude bits plus sign). A total of 20 arithmetic and logical decision operations are available for program instructions.

The equipment located with the RW-300 computer in the Control Center is shown in Figures 3 and 4 and includes a Flexowriter, X-Y plotter, a Master Control Console, and the servo-system electronic equipment and instrumentation equipment. The Flexowriter serves as the primary input-output device for the computer and includes a paper tape reader and punch unit. The tape reader is used to load the instructions into the computer memory and to provide the test parameters and variations for the program. The Flexowriter is also used for tabulation of test results.

Complementing the Flexowriter as an output device is a Mosely 2A X-Y plotter. The plotter is computer-controlled and can plot up to six separate Bode diagrams describing the servo system as the test is being conducted. The Bode plots may describe the various loop or component frequency responses, or, by the assignment of each plotter symbol to a separate driving function amplitude, may describe the amplitude-dependent nonlinearity of the servo system.

The Master Control Console houses the intercom switching and the television camera switching. A Hewlett-Packard 202A function generator, modified for digital frequency and amplitude control, is also located in the console. The function generator is used to provide the driving functions for the frequency response tests on the servo systems. Digital drawers, which contain the relays which effect the digital outputs to the Flexowriter, X-Y plotter, and function generator complete the Master Control Console.

The four-bay electronic equipment console (Figure 4) houses all the servo system electronics equipment. Many amplifier, filter, and compensation network combinations are located in the console. Digital outputs from the computer are used to form a specific configuration of these elements for the particular servo system hardware being tested. Also located in the console are the strain gauges and thermocouple reference junctions which make available computer monitoring of "g"-loads, pressures, flows, temperatures, etc., at the test site during the test.

SYSTEM OPERATION AND COMPUTER PROGRAM

The capability of the computer to read punch-coded paper tape from the Flexowriter concurrently with the execution of the test provides much of the versatility of the automatic test system. The sequence of tests to be conducted, the specific driving functions to be applied, and the data to be obtained are coded and listed on paper tape. This information is read as required by the computer during the execution of the test.

The basic block diagram of the computer program is shown in Figure 5. The program is divided into six regions, each of which performs a specific function for the test. Entrance into a region is only by a code read from the tape identifying that region. When the test is started, Region I is entered. Here the time of day is recorded, and the test is assigned an identification number, also printed on the result sheet. The exit from Region I is Region II, where a digital input from the tape reader is performed. The code read from the tape is interpreted, and the program exits to the particular region identified by the code. When the instructions of the region have been performed, the program immediately returns to Region II to read another region code from the tape. Thus the regions are entered in the order which they are coded on the tape.

The three basic types of tests available are (1) Equipment Calibration, (2) System Frequency Response, and (3) System Stability Test. The regions which perform these tests are identified as C, R, and T respectively. In addition, for the equipment calibration and frequency response tests, there are two regions which operate to set up a required driving function. These two regions are identified by 'F' and 'A'. In Region F, the frequency of the driving function is read from the tape and, by means of the digital outputs, set on the function generator. The desired driving function amplitude is, in a like manner, set in Region A.

A typical test tape format is shown in Figure 6. The first character in each line identifies a region. The digits following the region codes contain the information required by the region to complete the operations in that region. The digits following the C and R codes (calibration, frequency response) are used to instruct the program (1) what output ratios to compute, and (2) whether these gain and phase results should be printed, plotted, or both printed and plotted. The digits following the F code (frequency) are the octal representation for a particular driving function frequency. The driving function amplitudes follow the A (amplitude) region codes. The digits following the T (stability test) contain the information necessary for the stability test. These digits contain codes for (1) the gain settings of the amplifiers, (2) the compensation networks to be digitally inserted into the

various loops, and (3) the results to be printed at the completion of the test. The S code is punched at the end of each tape and is interpreted by the program as "test completed." The computer stops operation upon receipt of this code, and the operator may switch to another test location to begin another test on a different servo system.

For a frequency response test on the servo system or an electronic equipment calibration, then, three regions must be listed. (R, F's and A's or C, F's and A's). For a stability test, only region T is entered. The test engineer lists the codes necessary to conduct the specific test desired. The digits following each region code are obtained from a set of tables. A tape is punched containing these codes in proper sequence and placed in the Flexowriter tape reader. After switching to the desired test site in the laboratory, computer operation is begun. The test is then performed automatically, the desired results are printed and/or plotted, and the computer ceases operation when the test is complete.

DATA ACQUISITION AND MAJOR CALCULATIONS

After the driving function has been selected and applied to the system, the data from which the system response is calculated are obtained in the following manner. Programmed digital outputs from the computer connect four lines from the servo system through scaling amplifiers to the input of the analog-to-digital converter. The scaling amplifiers are controlled by the computer to scale the voltages to the most accurate range for conversion. Upon command, these four signals are sampled in sequence at the rate of 3840 samples/second or 960 samples/line/second. The digital values of these samples are stored in a prescribed sequence on the reserved section of the magnetic drum memory. The converter is bipolar, and converts voltages ranging to ±10.23 volts with a resolution of ±10 millivolts.

Since the section of the RW-300 memory reserved for the storage of analog data contains 1024 words, each of the four signals is sampled 256 times during one conversion cycle. From these samples, the amplitude and relative phase of each signal are determined. The method used is that provided by Fourier analysis. The real and imaginary components of the fundamental of each signal are given by Equations (1) and (2).

$$A = \frac{1}{\pi} \int_0^{2\pi} f(t) \sin wt \, dt \qquad (1)$$

$$B = \frac{1}{\pi} \int_0^{2\pi} f(t) \cos wt \, dt \qquad (2)$$

Where:

A = Real part of fundamental of f(t)
B = Imaginary part of fundamental of f(t)
W = Radian fundamental frequency

Because each signal f(t) is represented in discrete samples, it is necessary to approximate these integrations by numerical methods. The trapazoidal method of numerical integration is used, and the above integrals are approximated by Equations (3) and (4).

$$A \simeq \frac{2}{N}\left[\frac{S_o}{2}\sin(o) + \sum_{k=1}^{N-1} S_k \sin k\Delta\theta + \frac{S_N}{2}\sin N\right] \qquad (3)$$

$$B \simeq \frac{2}{N}\left[\frac{S_o}{2}\cos(o) + \sum_{k=1}^{N-1} S_k \cos k\Delta\theta + \frac{S_N}{2}\cos N\right] \qquad (4)$$

Where:

$S_o, S_1, \ldots S_k \ldots S_n$ = Samples of f (t)
N = Total number of samples
$\Delta\theta$ = Angle between samples S_k and S_{k+1}

The integrations of Equations (1) and (2), and hence the above summations, must be performed over an integer number of cycle of f(t). Since the sampling rate is fixed, the number of samples N is varied with frequency so that, as nearly as possible, an integer number of samples are used to represent an integer number of periods of f(t). It has been found that for $N \geq 30$, sufficient accuracy is maintained. At least thirty samples of f(t) are therefore used in the above summations.

Rectangular-to-polar conversion of the results of Equations (3) and (4) provide the amplitude and phase angle for each signal as given in Equation (5).

$$f(t) = C\angle\phi \qquad (5)$$

Where:

$$\phi = \tan^{-1}\frac{B}{A}$$

$$C = \frac{B}{\sin\phi}$$

After each of the four input signals have been represented in this manner, the desired amplitude ratios and phase relationships are calculated. The logarithms of the amplitude ratios provide the gain in decibels. The component, loop, or system gain and phase relationships are then printed with the Flexowriter and/or plotted with the X-Y plotter. The program is then ready to read the coded tape to obtain the next requested driving signal.

An example of a typical result sheet is shown in Figure 7. This particular test began with an equipment calibration. Under CALIBRATION are listed the static test parameters. These include the servo-amplifier quiescent currents, the servo-amplifier gain, and the feedback transducer excitation voltages. A brief frequency response test on the electronic equipment in the feedback loops was then conducted. Under FREQUENCY RESPONSE, the component, loop, and system gain and phase relationships are tabulated for various driving functions. Under each column is listed the gain in decibels followed by the phase shift in degrees.

Plotted on the X-Y plotter during this test (Figure 8) was the closed-loop system frequency response. Each driving function amplitude was assigned a different symbol on the plotter. The spread observed on the plot is therefore a result of the amplitude-dependent nonlinearity of the servo system.

SUMMARY

The procedure used by the test engineer in conducting a dynamic test on the servo system with the automatic test system is as follows. First, the hardware to be tested is set up at one of the remote sites in the laboratory. The loading (inertia, spring, damping, etc.) and the environmental conditions for the servo system are arranged. A test tape, listing the codes for the particular tests desired, is punched with the Flexowriter punch unit. This paper tape is placed in the Flexowriter tape reader and the automatic test system is switched to the remote site where the servo system is located. Computer operation is then begun. Without manual intervention, the computer program reads the punch-coded tape as the information is needed, executes the operations instructed in each program region listed on the tape, and prints and/or plots the results of the test as they are obtained. When the test is complete as indicated by a code on the test tape, the computer halts and the engineer has the desired tabulated and plotted transfer functions at hand. Another test engineer, with an entirely different servo system located at another remote site in the laboratory, may then switch the automatic test system to his location and follow the same procedure.

The justification of this serial test procedure in replacing parallel manual testing has been achieved. A typical dynamic servo system test conducted manually required up to two hours to obtain sufficient data with which to evaluate the system. An additional two hours were usually required to calculate the results and plot the desired transfer functions. A similar test conducted with the automatic test system now takes from 10 to 20 minutes (depending upon the extent of the test). Since the results are obtained in plotted form, the entire test is accomplished in only a small fraction of the time previously required. The many required tests can therefore be conducted serially at a more rapid rate than was previously possible using the parallel manual method. The savings resulting from the elimination of duplicate test equipment will be considerable. A by-product of the time savings is the reduction of servo system component wear. Because the servo system is being driven only a fraction of the previous time, wear and the resulting changes in operating characteristics have been minimized. The previous expense of maintaining several prototypes of each component has thus been eliminated.

In addition, the accumulative errors accrued during a manually performed test were estimated to provide results accurate to only ± 10%. The digital computer provides these test results accurate to ± 1%. Thus more confidence can be placed on the test results, and more accurate evaluations of the system performance can be made.

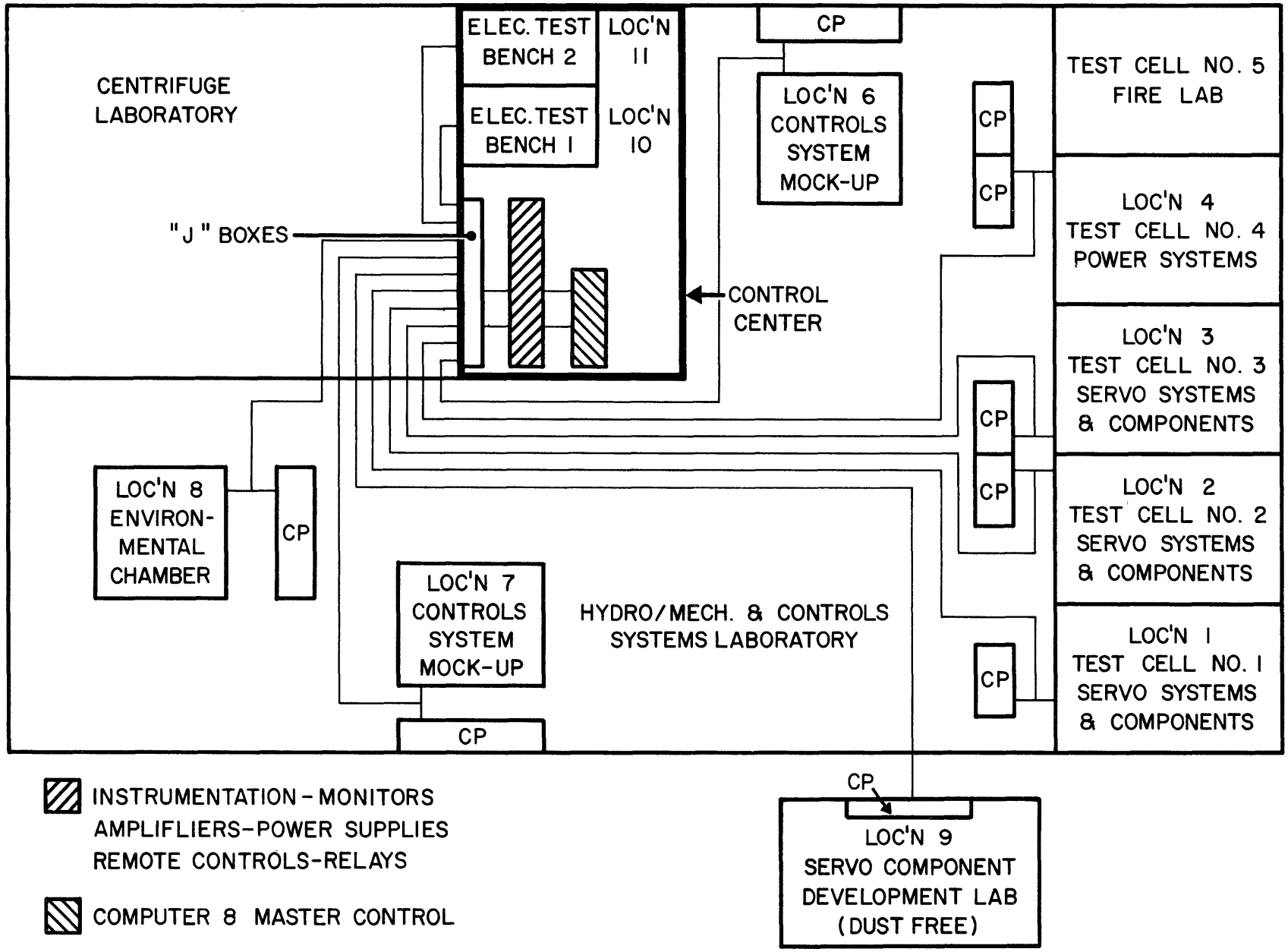


Fig. 1. Systems Laboratory Layout

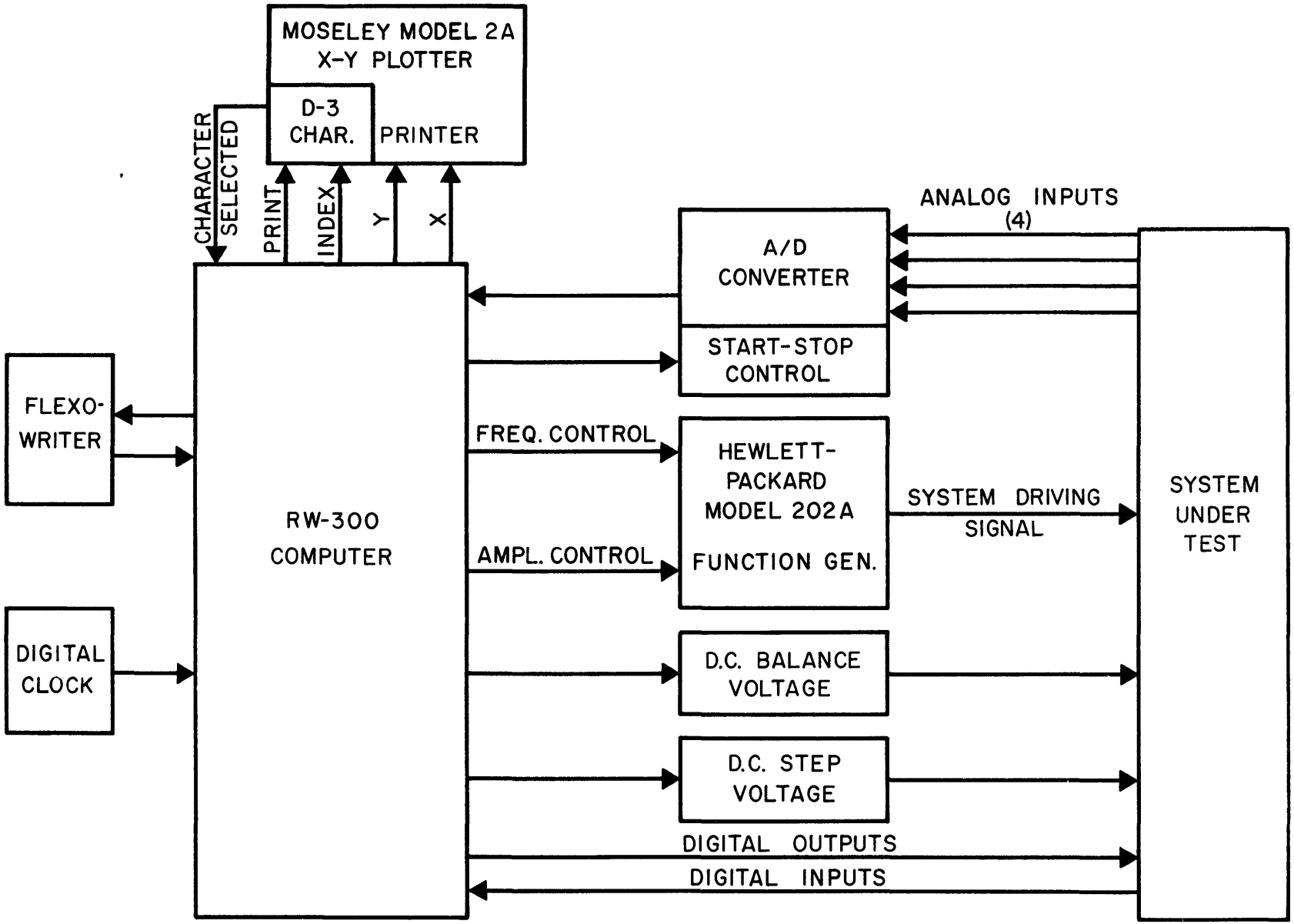


Fig. 2. RW-300 Computer Input-Output System

Fig. 3. Systems Laboratory Control Center

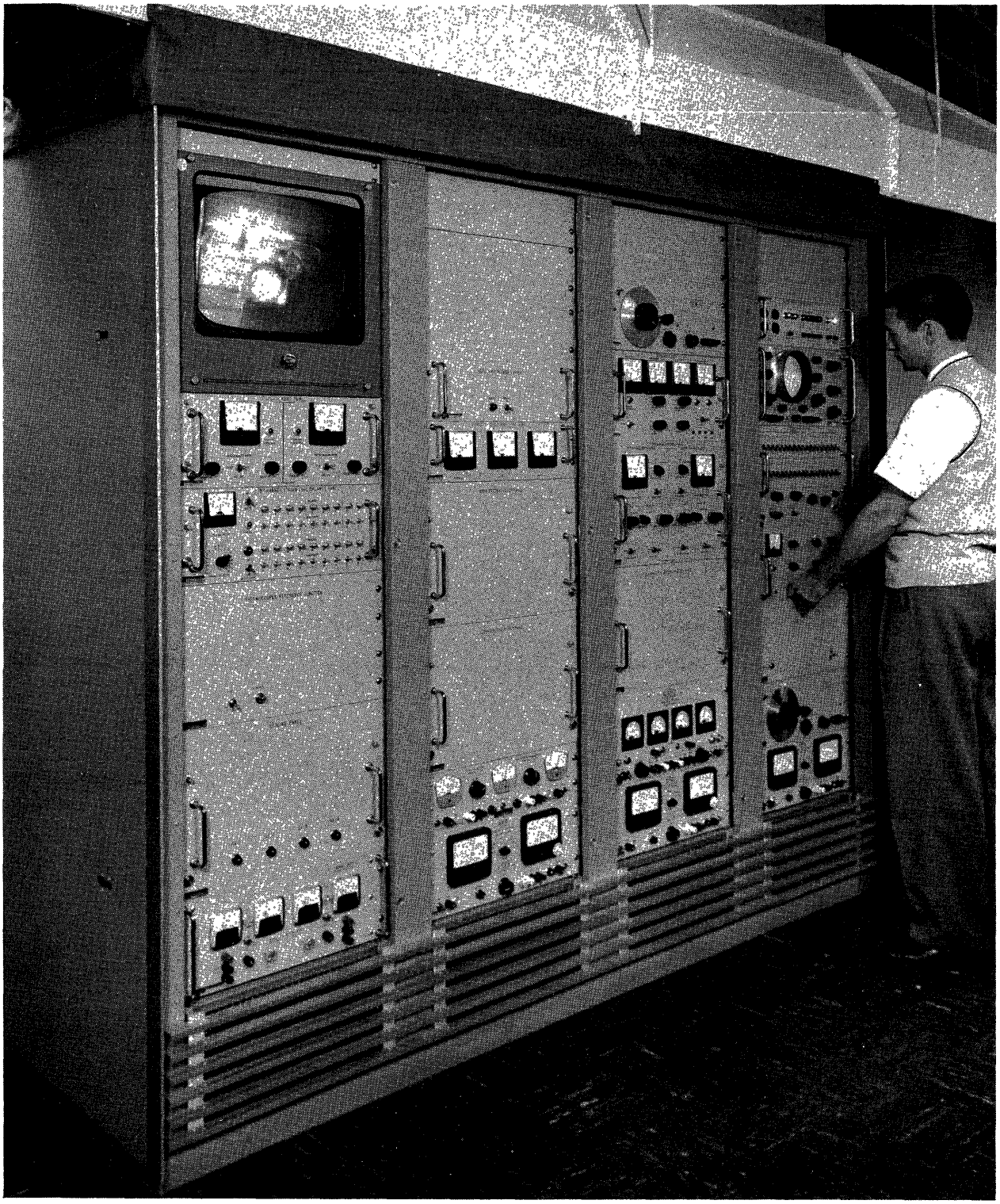


Fig. 4. Servo System Electronic Equipment and Instrumentation Console

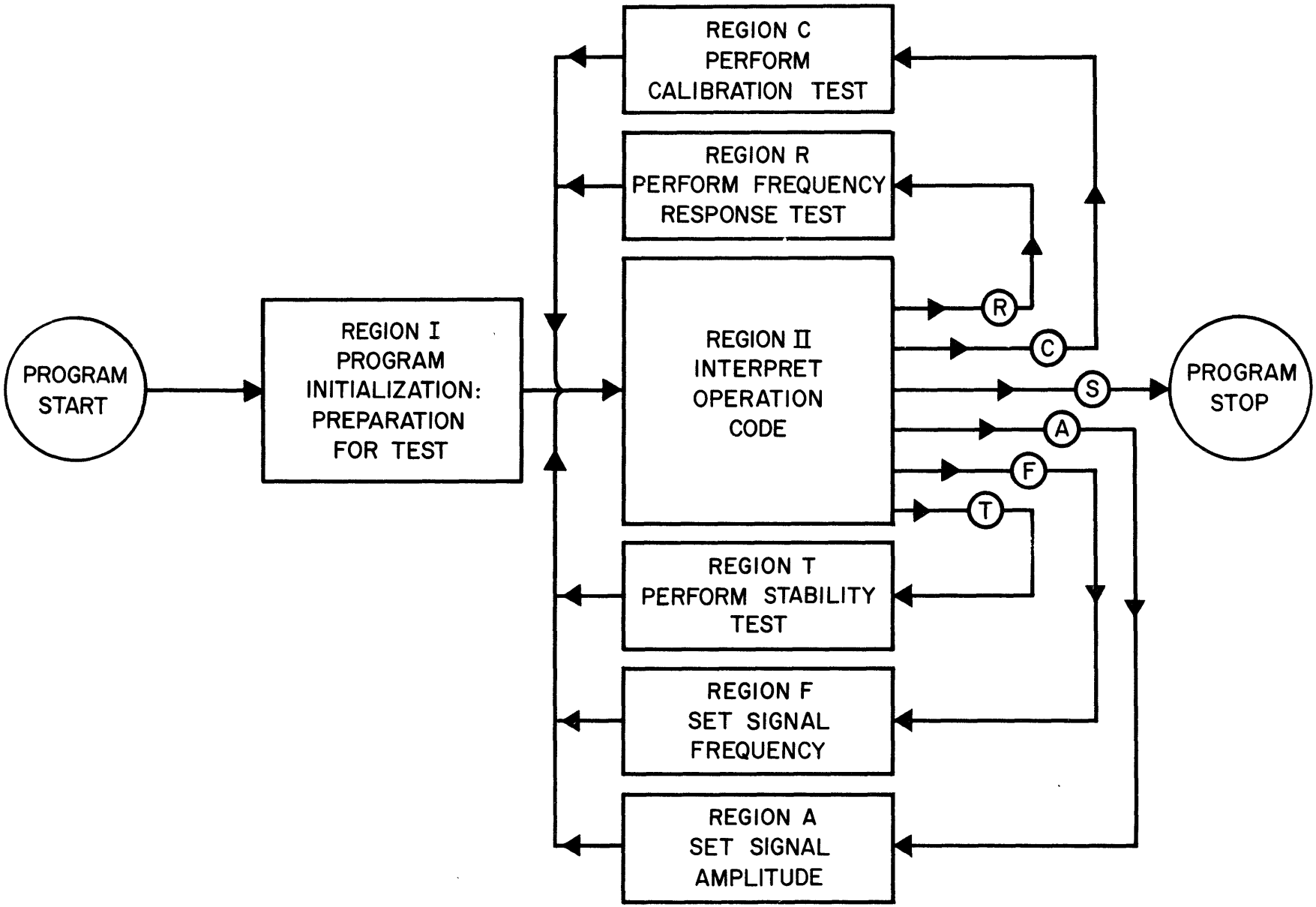


Fig. 5. Basic Program Block Diagram

```
C212431410.
F001300
A000100
A000200
   .
   .
   .
F006300
A000100
   .
   .
   .
R432210310.
F000200
A000120
A000160
   .
   .
   .
T100036043172.
S
```

Fig. 6. Test Tape Format

10-20-60 1022 AM
RUN NO. 3446

CALIBRATION
I1 211.9 MA
I2 209.6 MA
AMP. GAIN 100.2 MA/V
BUZZ 20.37 V
EX1 10.31 V
EX2 19.92 V

FREQ.	AMPL.	VFAA/VCAL	VFAV/VCAL	E2/VCAL
3.985	4.925	-0.066 -0.656	-0.027 -0.890	0.097 -0.750
49.92	4.839	-0.101 -1.507	-0.007 -2.277	0.085 -0.554
202.2	4.796	-0.031 -7.566	-0.074 -8.296	0.171 -0.031

FREQUENCY RESPONSE

FREQ.	AMPL.	VFB2/E1	VFB2/E2	VFB1/E1	VFB1/E2
3.960	0.980	-0.156 -190.6	13.43 -86.11	-6.925 -101.3	6.664 -356.7
	2.007	-0.113 -191.1	14.97 -94.12	-7.089 -100.9	8.003 -3.917
	3.980	0.011 -188.0	16.42 -95.92	-8.015 -100.1	8.390 -7.949
	6.019	0.034 -188.1	17.12 -97.25	-9.496 -98.64	7.574 -7.796
	8.097	0.066 -187.9	17.27 -95.73	-11.20 -98.94	5.996 -6.785
6.011	0.980	-0.269 -195.7	11.10 -92.75	-10.58 -106.6	0.789 -3.695
	2.042	0.031 -194.8	12.14 -96.31	-10.51 -106.1	1.593 -7.652
	3.960	-0.003 -192.5	13.22 -97.19	-11.36 -105.0	1.859 -9.796
	5.949	0.078 -191.7	14.04 -98.31	-12.88 -104.6	1.082 -11.19
	7.976	0.066 -191.2	14.44 -97.80	-14.73 -103.8	-0.351 -10.44
8.046	0.992	-0.265 -200.4	8.718 -96.71	-12.95 -111.1	-3.972 -7.503
	2.027	-0.101 -198.0	9.886 -97.72	-13.02 -110.6	-3.027 -10.31
	3.941	0.058 -195.8	11.17 -100.3	-13.90 -109.2	-2.785 -13.76
	5.914	0.074 -194.4	12.03 -98.86	-15.51 -108.3	-3.546 -12.75
	7.898	0.097 -193.8	12.30 -99.06	-17.27 -107.4	-5.070 -12.67
12.03	1.007	-0.324 -209.0	5.535 -99.85	-16.50 -123.5	-10.64 -14.30
	2.035	-0.164 -206.1	6.878 -101.0	-16.53 -122.8	-9.492 -17.72
	3.914	0.144 -202.4	8.285 -102.5	-17.39 -118.6	-9.250 -18.75
	5.851	0.132 -200.3	9.132 -103.5	-18.99 -115.2	-9.992 -18.39
	7.875	0.140 -199.1	9.531 -101.9	-20.77 -113.4	-11.38 -16.37
15.95	0.992	-0.562 -217.6	3.343 -104.2	-18.98 -134.9	-15.07 -21.50
	2.003	-0.023 -213.9	4.636 -105.5	-18.92 -130.5	-14.26 -22.15
	3.945	0.246 -207.9	6.347 -107.3	-19.84 -123.4	-13.75 -22.86
	5.867	0.308 -205.1	7.304 -107.1	-21.48 -120.3	-14.48 -22.34
	7.863	0.339 -202.9	7.914 -106.5	-23.28 -116.8	-15.70 -20.46
20.07	1.000	-0.792 -227.9	1.296 -108.1	-21.78 -147.7	-19.69 -27.94
	2.027	-0.257 -220.7	2.964 -109.7	-20.49 -141.1	-17.26 -30.07
	3.917	0.382 -214.0	4.632 -110.2	-21.62 -131.5	-17.37 -27.74
	5.890	0.503 -209.9	5.898 -111.1	-23.42 -125.6	-18.02 -26.89
	7.851	0.562 -207.9	6.437 -111.3	-25.21 -123.5	-19.34 -26.84
25.05	0.972	-0.968 -239.3	-0.464 -114.7	-23.04 -156.2	-22.53 -31.58
	2.003	0.003 -230.8	1.386 -115.6	-22.58 -150.1	-21.20 -34.89
	3.941	0.679 -220.4	3.394 -116.3	-23.67 -138.8	-20.95 -34.76
	5.949	0.902 -216.0	4.425 -116.9	-25.25 -131.4	-21.73 -32.34
	7.824	1.097 -214.0	4.957 -118.6	-27.21 -130.9	-23.35 -35.46
30.01	0.960	-2.105 -250.9	-2.476 -115.5	-23.11 -180.4	-23.48 -45.04
	2.015	-0.839 -241.2	-0.742 -116.4	-24.27 -165.3	-24.17 -40.63
	3.898	0.488 -225.4	2.328 -116.1	-24.80 -149.9	-22.96 -40.57
	6.250	1.375 -219.8	3.789 -122.2	-27.78 -141.5	-25.36 -43.95
	8.382	1.781 -220.0	3.785 -125.8	-29.01 -138.0	-27.01 -43.92
40.04	0.960	-3.367 -267.8	-4.972 -121.8	-31.74 -201.6	-33.35 -55.57
	1.972	-0.945 -261.9	-3.050 -128.3	-28.04 -184.9	-30.15 -51.35
	4.156	1.082 -241.5	0.171 -126.2	-27.03 -164.9	-27.94 -49.69
	5.886	2.332 -238.9	0.980 -132.9	-29.35 -157.6	-30.70 -51.74
	8.246	2.542 -248.9	-0.164 -136.8	-32.01 -169.7	-34.73 -57.57
49.99	0.988	-5.238 -288.8	-7.476 -132.4	-35.57 -240.5	-37.81 -84.14
	2.050	-2.339 -277.0	-4.921 -132.2	-30.32 -204.1	-32.91 -59.28
	4.042	1.445 -259.9	-1.601 -135.6	-29.00 -181.3	-32.05 -57.13
	5.953	1.957 -265.7	-1.886 -139.5	-30.81 -186.1	-34.66 -59.94
	7.765	1.707 -278.1	-2.882 -143.8	-33.51 -197.2	-38.10 -62.92
59.74	0.980	-7.261 -302.7	-9.488 -139.5	-34.01 -321.9	-36.23 -158.7
	2.042	-4.738 -295.9	-7.480 -138.9	-32.99 -235.3	-35.73 -78.33
	4.027	0.500 -280.9	-3.585 -142.3	-30.53 -201.6	-34.62 -63.10
	5.972	1.183 -286.1	-3.582 -145.6	-33.48 -208.5	-38.25 -68.07
	7.898	0.171 -297.2	-4.699 -148.5	-35.13 -216.8	-40.00 -68.16
79.91	0.984	-11.30 -328.4	-13.25 -155.3	-47.89 -170.5	-49.83 -357.3
	2.015	-9.414 -319.3	-11.66 -149.6	-40.25 -261.1	-42.51 -91.36
	4.007	-4.667 -313.0	-8.082 -149.5	-37.44 -243.7	-40.85 -80.19
	5.964	-2.289 -313.8	-6.593 -153.4	-36.00 -238.5	-40.31 -78.05
	7.933	-2.828 -319.1	-7.082 -155.7	-37.94 -243.4	-42.19 -80.01
101.9	0.984	-17.05 -336.6	-18.12 -159.5	-42.03 -270.7	-43.10 -93.62
	2.011	-14.63 -335.3	-16.01 -160.0	-52.42 -266.2	-53.80 -90.91
	3.937	-10.35 -333.7	-12.54 -159.3	-46.17 -267.5	-48.37 -93.13
	6.015	-7.246 -338.6	-10.30 -164.9	-41.08 -271.0	-44.14 -97.35
	8.019	-6.816 -338.8	-10.03 -164.8	-40.94 -267.5	-44.16 -93.60

Fig. 7. Typical Tabulated Result Sheet

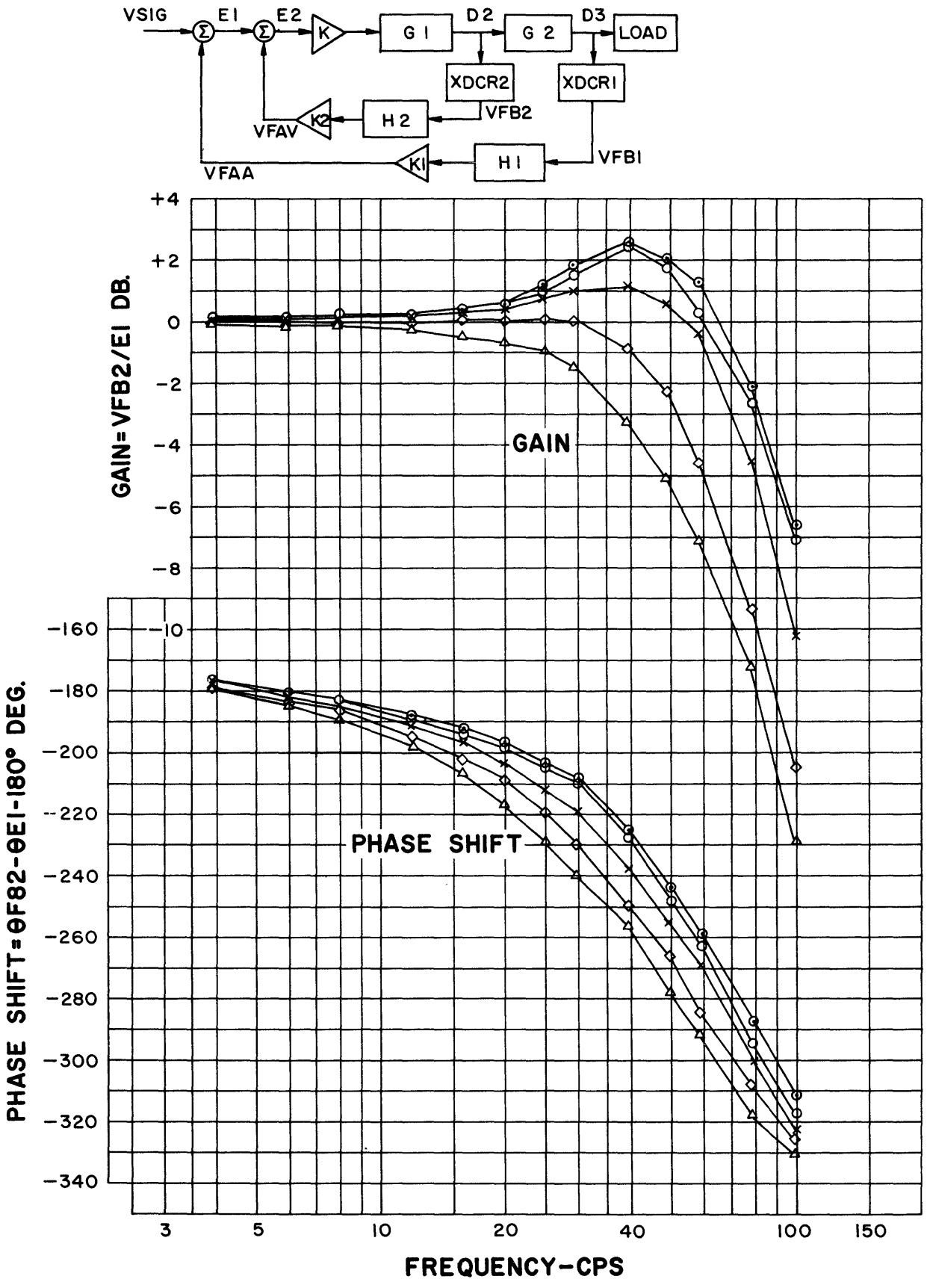


Fig. 8. Typical Plotted Result Sheet

HOT-WIRE ANEMOMETER PAPER TAPE READER

John H. Jory
Soroban Engineering, Inc.
Melbourne, Florida

Summary

The Hot-Wire Anemometer Paper Tape Reader was conceived as a relatively simple apparatus to serve as a high reliability device such as required in peripheral computation equipment. The hot-wire anemometer principle has been employed extensively in the past for the study of transient air flow phenomena in compressors and turbines. The principle of operation concerns an electrically heated wire used to detect a change in air velocity in its immediate vicinity by observing its change in temperature and consequent change in resistance. Using this principle to read perforated members such as punched paper tape at high speeds offers a number of distinct advantages over conventional reading methods.

Methods of Reading Paper Tape

At present there are available three general types of paper tape reading mechanisms. Brush type readers employ brushes which rest on one side of the paper tape and pass through perforations to engage contacts disposed on the opposite side of the tape. Sensing pin type readers mechanically insert pins through perforations occurring in the tape and thus accomplish the reading function. Photoelectric readers employ a beam of light directed toward a photocell which is interrupted at all times except when perforations are disposed between the photocell and the light source. The two former members are relatively slow since the inertia of the sensing members limit the rate at which perforations may be sensed. The photoelectric method is materially more rapid than the other two methods but suffers from difficulties arising from the presence of dirt and paper chaff in the region of the photocells. More particularly, dust and bits of paper from the perforated member accumulate around the photocells and limit the amount of light reaching the photo-sensitive element. Unless the photocells are regularly cleaned the amplitude of signals decreases to a point where the circuits and system become unreliable. A further problem arises in the reading of oiled tapes which are translucent, causing reflections from the tape to appear as point sources at the tape, introducing cross-talk between circuits. In order to overcome these difficulties specialized techniques must be employed. Also inasmuch as oiled paper tapes are the most commonly used in the field due to their reducing wear on punch pins, photoelectric readers are limited in their application to paper tape handling.

Hot-Wire Reader

The Hot-Wire Reader operates by allowing air under pressure to pass through perforations in punched paper tape and be directed over electrically heated wire elements which are thereby cooled, causing their resistance to change and thus indicating the presence of a perforation in the tape. In the most advantageous interpretation of this reading scheme the paper tape is situated between the following sub-assemblies:

1. A plenum chamber with an arcuate surface in which a slit is cut which bears on the tape. This chamber is kept at a low pressure by means of a suitable vacuum source.

2. A sensor plate having a small window opposite each channel position of the tape being read, the windows being aligned with the opposed slit in the plenum chamber.

A small coil of wire is situated within each window in the sensor plate and these sensing elements change their temperature and resistance when a perforation in the tape passes between them and the slit in the plenum chamber. This hot-wire reading technique inherently overcomes the difficulties encountered with photoelectric reading. Since there is sporatic air flow through the reading station, dust, dirt, and lint are automatically removed and deposited in the pump filter. The reader is capable of reading optically transparent, translucent, or opaque media. Moreover, the reader is simple and inexpensive, utilizing readily available amplifiers and hot wires. The reading speeds attainable are limited only by the mechanical transport mechanism and the quality of the amplifiers used. In addition changes in ambient temperature do not affect reading even though changes in hot wire temperature are being detected since the temperature of

hot-wires is maintained at about 400°C. which is quite high with respect to ambient. The amplifiers adjust their drive continuously to maintain constant resistance in the hot-wire which is compared with a fixed resistor in a bridge circuit.

Hot-Wire Elements. The hot-wire elements are composed of 15 or 20 turns of nickel-iron alloy wire with a high temperature coefficient of resistance. The coils are 0.001 inch diameter wire wound on a 0.005 inch mandrel. Hot-wire elements themselves have relatively high thermal time constants, and so a system which allows the hot-wire to change temperature in response to the initiation and termination of air flow only would be slow in comparison to photo-electric readers. By employing amplifiers with sufficient feedback the temperatures of the hot-wires are maintained essentially constant, and a resistance change merely sufficient to generate an error signal occurs. In addition the hot wires are wound in coils as opposed to the conventional straight wire elements. With straight wire elements the thermal inertia of the end supports is found to affect the response of the system. Thus by employing a coiled wire, the major portion of the wire is substantially removed from the end supports so that their thermal inertia does not injure the response.

Flexibility of Reading Technique. Either continuous or discontinuous reading may be employed with the hot-wire reader and either AC or DC circuits may be used as amplifiers although there is a well known tendency for DC circuits to drift. Also if discontinous reading is employed a code may remain stationary over the reading station thereby allowing the device to act as a temporary storage register until the equipment again begins to transport. If the device stops with a web of the tape over the reading station there will be no output signal at this time and readings will be obtained while transporting the tape. If the hot-wire sensing elements are placed within the plenum chamber and a low pressure maintained there, paper tape could be pulled over the arcuate surface of the chamber and edge guided only, not requiring any sort of clamping device on one side of the tape. In this configuration the chamber would have holes rather than a slit so that each channel could be sensed from inside the chamber. This would eliminate any threading inconvenience when loading the reader with tape.

One extremely important aspect of the Hot-Wire Reader is the lack of the need for sophisticated components in the construction and operation. The operating air pressure required is approximately one inch of water so that extremely small blowers and motors may be employed. Paper tape channels are on one-tenth inch centers allowing ample space for mounting of hot-wires. Due to the freedom from complexity of this apparatus, it is an extremely reliable device, which concept is gaining more and more importance in the data handling field.

USE OF A DIGITAL/ANALOG ARITHMETIC UNIT WITHIN A DIGITAL COMPUTER

Donald Wortzman
IBM Corporation
Yorktown Heights, New York

Summary

A novel approach to arithmetic operations in digital computers is described which combines digital and analog techniques. This is accomplished by using parallel-serial interconnections of digital/analog converters. The interconnections are under stored program control and are effected by selecting multiplexers which route the analog signals to perform various arithmetic operations. The final analog signal, which is the result of the computation is converted to digital form by means of an analog/digital converter.

Because of the extreme parallel nature of D/A Arithmetic, high computational speeds are attained, although all the circuitry is operating slowly by present digital computer standards. Using present techniques the analog nature of the computation limits the accuracy to four or five significant decimal figures. Therefore, the D/A Arithmetic Unit could not replace the arithmetic unit presently in digital computers, but could be used to solve problems or parts of problems that do not require extreme accuracy.

Digital/Analog Arithmetic

Introduction

A computer is usually classified as being either analog or digital. Each has its advantages and disadvantages, when handling mathematical problems. In a previous paper[1], it was shown how digital and analog techniques could be combined to synthesize various analog type building blocks having accuracies not obtainable with conventional analog techniques.

This paper suggests a method of combining digital and analog techniques in order to perform the arithmetic operations in digital computers. The techniques offer the potentiality of extremely high computational speed at low cost.

Arithmetic operations in the arithmetic unit described in this paper are performed without the use of conventional adder circuitry as found in most stored program computers. This is done by routing analog signals, by means of multiplexers, onto the reference voltage inputs of D/A Converters, summing the analog signals thus formed, and finally converting the result to digital.

Before beginning the discussion of the D/A Arithmetic Unit, a brief discussion of D/A conversion, building blocks of D/A arithmetic, and several interconnections of them will be given.

Digital to Analog Conversion

Figure 1 depicts a unipolar, three binary bit, digital to analog converter, D_1, D_2, and D_3 are single-pole double-throw switches which can be thrown independently to zero volts or V_r volts; zero being a logical 0 and V_r volts being a logical 1. If the output voltage is calculated as a function of D_1, D_2, and D_3, equation 1 results:

$$V_{oc} = \left[4D_1 + 2D_2 + D_3\right] \frac{V_r}{8} \quad . \qquad (1)$$

It should be noted that the bracketed factor in equation (1) is the definition of a binary number, where D_1 is the most significant bit, D_2 is the next most significant bit, and D_3 is the least significant bit. Further, equation (1) illustrates that the open circuit voltage is not only proportional to the digital input but is also proportional to the reference voltage, V_r.

Digital/Analog Building Blocks

Digital/analog building blocks have both digital and analog inputs and outputs; to provide clarity the following rules are followed in this report:

1. All digital quantities enter and leave the top of the block.
2. All analog quantities enter and leave the sides of the block.

Figure 2, the D/A Converter, is the block representation of Figure 1. This block is important since it is through this that multiplication is performed. A requirement of the D/A Converter is that it present a small load to the voltage reference. The current that flows from the voltage reference must first pass through the the multiplexers, therefore a large current would produce a large voltage drop across the multiplexer's "on" impedance.

Figure 3 is a summing amplifier. It is used for both impedance matching and summing of the analog signals. Since it is required to have little dc drift, it would most likely be chopper stabilized.

The multiplexer, Figure 4, is one of the most important blocks, since it is through multiplexing that the different arithmetic operations are performed. It is required to have a low "on" impedance and a moderately high "off" impedance .

The circuit of Figure 5 is a variable gain amplifier. This circuit is similar to the summing amplifier except that the feedback can be varied by means of the digitally controlled D/A Converter, which can change the gain to any of three values, one-tenth, one or ten. This scaling helps conserve the accuracy of the D/A Arithmetic Unit.

The comparator, Figure 6, is the last of the building blocks that will be described. A positive current flows into A and a negative current flows out of B. The digital output D indicates which current is larger in magnitude. This circuit is used in converting the analog result of the arithmetic computation to digital form.

Interconnection of Building Blocks

In Figure 7, the digital output of the comparator controls the logic which strives to make B_1 and B_2 equal.

Under this condition:

$$A_1 \cdot D_1 = A_2 \cdot D_2 \qquad (2)$$

If A_1, A_2 and D_2 are preset and D_1 is varied until $/B_1/=/B_2/$, then equation 2 can be rewritten as:

$$D_1 = \frac{A_2}{A_1} D_2 \qquad (3)$$

If on the other hand, A_1, A_2 and D_1 are preset and D_2 is varied until $B_1 = B_2$, then equation 2 becomes:

$$D_2 = \frac{A_1}{A_2} D_1 \qquad (4)$$

The significance of equations 3 and 4 is that depending on whether D_1 or D_2 is preset, the function or its reciprocal can be digitized.

In Figure 8, C_1 is equal to the negative product of A_1 and D_1 or

$$C_1 = -A_1 \cdot D_1 \qquad (5)$$

Similarly C_2 is equal to the negative product of A_2 and D_2 or

$$C_2 = -A_2 \cdot D_2 \qquad (6)$$

The voltage B is equal to the negative of the sum of C_1 and C_2 or

$$B = A_1 \cdot D_1 + A_2 \cdot D_2 \qquad (7)$$

In Figure 9, the output voltage B is equal to the product of U, D_1 and D_2. U is the normal reference voltage and is assigned the value unity,

therefore:

$$B = U \cdot D_1 \cdot D_2 = D_1 \cdot D_2 \qquad (8)$$

Digital/Analog Arithmetic Unit

Thus far, the digital/analog building blocks and some simple interconnections of them have been explained. Figure 10 is the digital/analog arithmetic unit. It consists of three sections.

Section I performs the arithmetic operations. It contains 16 D/A converters, 24 negative unity gain amplifiers and 72 multiplexers. It should be remembered that with various interconnections of D/A converters and amplifiers many arithmetic operations can be performed. However, with as many as 16 D/A converters, it is unlikely that any particular arithmetic operation would have many applications. The multiplexers reroute the analog signals so that many different arithmetic operations can be performed. In other words, by means of multiplexing, the interconnection of the D/A converters can be altered, thereby changing the programmed arithmetic operations within a few microseconds.

Section II is a digitally controlled variable gain amplifier. It adjusts the gain so that the input to Section III is as large as allowable. This is done because Section III contributes the largest part of the total error, so that if the gain of the variable gain amplifier is large, the relative error due to Section III is small.

Section III performs the analog to digital conversion. It can perform one of two functions. Either it converts the value of the input analog signal or it converts the reciprocal of the input analog signal. If the input voltage is positive, then the input is used as reference for D/A_{18}, and -U is used as reference for D/A_{19}. Recalling from Figure 7 that depending on which of F_2 or F_3, Figure 10, is preset and which is varied until the inputs to the comparators are equal will determine whether the function or its reciprocal is digitized. If the input is negative, then the input would be used as reference for D/A_{19} and +U would be used as reference for D/A_{18}. The polarity indicator is used to make this decision.

In summary, the arithmetic operations are performed in Section I, and Section II and III convert the output of Section I into digital form.

To illustrate the flexibility of digital/analog arithmetic two different problems are set up for solution. The first is the computation of the first eight terms of a Taylor series with arbitrary coefficients; the second is the multiplication of two eight by eight matrices.

Example One. Assume that it is desired to calculate e^x for arbitrary values of x. If e^x is expanded in a Taylor series the result is:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} \quad \ldots. \quad + \frac{x^7}{7!} \qquad (9)$$

If the proper multiplexers of Figure 10 are in the 1 state and all other's are in the 0 state, the interconnection illustrated by the heavier line in Figure 11 results.

In Figure 11, the output of A_3 is equal to x. It is the reference to D/A_5, therefore, the output of A_5 is proportional to x^2, and similarly the output of A_7 is proportional to x^3, and so on. The digital input to D/A_2 is 1, therefore, the voltage at B_1 is equal to 1. The digital input to D/A_4 is also 1. Because its reference is equal to x, the voltage at B_2 is equal to $1 \cdot x$ or just x. Similarly, the voltage at B_3 is equal to $x^2/2!$ and so on. The voltages of B_1, B_2 ...B_8 are all summed in the summing amplifier in Section II and its output is equal to:

$$1 + x + \frac{x^2}{2!} + \ldots. + \frac{x^7}{7!} \qquad (10)$$

or e^x to the accuracy of the 8 terms of the Taylor series. This voltage can be used as such, or it can be converted to a digital number in Section III. If the value of e^x for a different value of x is desired, the coefficients remain the same and x is changed. For values of x greater than 1, 8 terms of the Taylor series may not be sufficient. For this case additional terms can be calculated separately and the results added.

Example Two. The second example is the multiplication of two eight by eight matrices. In symbolic form:

$$(C) = (A)\ (B) \qquad (11)$$

where the ij^{th} entry of (C)

$$C_{ij} = a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} + a_{i3} \cdot b_{3j} \ldots$$

$$\ldots.. a_{i8} \cdot b_{8j} \qquad (12)$$

Figure 12 is the interconnection (heavy lines) that would perform this operation. The digital input to D/A_1 is a_{i1}, making the reference to D/A_2 proportional to a_{i1}. The digital input to D/A_2 is b_{1j} and therefore B_1 is a voltage proportional to $a_{i1} \cdot b_{1j}$. Similarly B_2 is proportional to $a_{i2} \cdot b_{2j}$ and so on. The voltage of B_1, B_2 .. B_8 are all summed in the summing amplifier in Section II and its output is proportional to:

$$a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} \cdots a_{i8} \cdot b_{8j} \qquad (13)$$

This voltage is converted to digital form in Section III. When multiplying two eight by eight matrices, 64 such multiplications must be performed. In order to obtain the $C_{i(j+1)}$ term, the "a" entriesstay the same and only the "b" entries must be changed. In other words, in order to get succeeding C entries only half the information must be changed. This of course is time saving. If larger than eight by eight matrices are to be multiplied it is done in parts, and the results are added.

Although it has not been shown, the sign is handled by digital means. This is easily done because the information that chooses the proper channels to be multiplexed, determines the flow of information. It turns out that this coupled with the mathematical rules for signs is enough to compute the sign of each term. Referring to Figure 10 again, it can be seen that if S_1 is multiplexed in and S_2 is multiplexed out, B_1 will be positive. If the reverse is true, B_1 will be negative. The digital logic handling the sign decides whether B_1 should be positive or negative and selects S_1 and S_2 accordingly. B_2, B_3 .....B_8 are handled in the same manner.

Another important point is that although only digital inputs were used in both examples, combinations of both digital and analog inputs could be operated on. In figures 10, 11, 12 multiplexers M4, M8, M12, etc. can be used for this purpose. The typical digital/analog arithmetic unit would have this flexibility.

The D/A Arithmetic Unit described in this report is just an idea, however a scaled-down model has been built, which at least demonstrates the feasibility of such a system.

The building blocks which make up the D/A Arithmetic Unit were originally designed for use in an A/D Converter. In order to test the A/D Converter a special tester was built. The tester would generate, by means of a D/A Converter, various voltages controlled by a fixed program and the A/D Converter would convert these voltages back to digital. This digital quantity would then be compared to the digital quantity in the tester. If they were different by more than a prescribed amount the tester would stop and an error indication would result. If the error was smaller than the prescribed amount the tester would generate a new voltage and the process would be repeated. If the system diagram of the converter-tester combination is referred to, Figure 13, it will be noticed that it has many of the component building blocks of the D/A Arithmetic Unit, and could be considered as a scaled model of it. The results of the tests made on the converter-tester combination not only demonstrated feasibility of the D/A Arithmetic Unit but gave indication of what could be expected in terms of speed and accuracy.

As far as speed goes the converter-tester combination was run at 100 usec. cycle period, a cycle consisting of generating a voltage in the tester, D/A_1, converting it back to digital in the converter, D/A_2 comparing it with the original digital input and then making the decision whether to generate another voltage or to stop the machine. Therefore, a D/A Arithmetic Unit which performs at a speed of 100 usec per operation is consistent with the actual performance of the scaled model.

In the data taken of the accuracy, the maximum error between the digital generated number and the digital output was less than $\pm$ .05 full scale, although the vast majority of errors fell within $\pm$ .02% full scale. This large difference between the maximum error and the vast majority of errors is to be expected in successive approximation type A/D Converters. The reason for this will not be discussed in this report. However, this error is quite small when one considers the number of contributing factors to it, for example, two D/A Converters, two independent references, one amplifier and one comparator. In the D/A Arithmetic Unit two separate references would not be used as was in the model since the reference is the most difficult to accurately control. There are two other facts about this test that are most encouraging. The first is that none of the semiconductor elements necessary for this high accuracy were specially selected for this purpose. Most of these semiconductors were randomly selected from normal digital computer lots. The second important fact is that many of the circuits used in this test were superceded by later designs, which would improve the operation considerably. These tests and those made on the individual circuits indicates that D/A and A/D conversions can be performed

with average accuracies of .001% of full scale and .01% of full scale respectively.

If these performance specifications are written in terms of Figure 10, then it appears practical that Section I could be made accurate to one part in 100 thousand of full scale and that Section III could be made accurate to one part in 10 thousand of full scale. The difference in accuracy between Section I and Section III is the main reason for requiring Section II. As an example assume that A·B is to be solved. A current is produced at the output of Section I accurate to one part in 100 thousand of full scale. Assume this produces an equivalent voltage at the output of Section II also accurate to one part in 100 thousand. If this voltage is numerically equal to 7.6328 volts, when it is converted in Section III four place accuracy is obtained, so that the voltage of 7.6328 volts will convert to 7.632 volts. In other words, the last place figure is lost. However, because Section II is a variable gain amplifier this last figure can be recovered. If now a slightly different problem is solved

$$A \cdot B - 7.6320 \qquad (14)$$

the voltage at the output of Section II will now become

$$7.6328 - 7.6320$$

or (15)

$$0.0008$$

Since Section III is only accurate to four places 0.0008 cannot be converted to digital in Section III, but if the gain of Section II is increased by a factor of 10, the output voltage of Section II would be 0.0080 and Section III could convert this voltage and obtain an eight in the last place. If the value of the first conversion 7.632 is added to the value of the second conversion $\frac{0.0080}{10}$, 7.6328 results. This is the answer desired. In other words, if Section I is accurate to five places, a five place answer can be obtained, by using techniques such as these.

Auxiliary Arithmetic Unit

In order to present a better understanding of how the D/A Arithmetic Unit might operate within a digital computer the entire auxiliary arithmetic unit of which the D/A Arithmetic Unit is a part will be described.

Figure 14 is the Auxiliary Arithmetic Unit (AAU). The inputs to AAU are from two sources. The digital information is from the digital computer (not shown) whereas the analog information is from external analog sources (also not shown), for example, strain gages, tachometers, thermocouples, pressure gages or any other device whose reading can more conveniently be converted to a voltage rather than directly to a digital quantity. The analog sources may be connected to the AAU through a bank of multiplexers. The output is a digital quantity which is the result of the computation and which generally goes back to the computer or possibly a remote D/A Converter which controls a part of a process by means of a voltage. The easiest way towards understanding of each of the building blocks in Figure 14 is to show the chronological order of events when solving a problem in AAU.

1. A "Load Information" command comes from the digital computer which tells AAU that the operation registers and the data registers are to be loaded. (Referring to Figure 10 and Figure 14, the data registers contain the digital information D_1, D_2 .....D_{16} and the operation registers select the operation or computation.) In order to permit this loading, the information gate associated with these registers is opened by the controls.

2. The digital information comes from the computer serial by word, each word containing a tag which tells whether it is an operation or data. The word is then gated into the proper register. This process is continued until the operation and all the data is loaded into the proper registers.

3. The "Load Information" line is lowered.

4. The signs, ψ , of the data D and the operation information M, N, V are applied to polarity decoder. The output S of the polarity decoder is applied as the digital control signals S_1, S_2, S_3, S_4, etc.

5. The operation information M, N, V is applied to their respective multiplexers which digitally controls the operation.

6. The digital information D is applied to the appropriate converters.

7. The "Start Computation" command is given.

8. The polarity output P which senses the polarity of the output of Section II, Figure 10, select T_1 and T4 or T3 and T4

accordingly. In Figure 11 and Figure 12 the polarity was negative so therefore T_2 and T_3 were multiplexed in.

9. The polarity output P together with ϕ, (which determines whether the function or its reciprocal is wanted) presets F_2 or F_3 in the manner previously described.

10. The scaling factor F_1 is selected. This can be done in several ways. One method is to have the polarity block of Figure 10 also give the approximate magnitude of the voltage at the output of Section II and then select the scaling of F_1 accordingly.

11. The answer appears in F_2 or F_3 depending on step 9. The information in F_1 is also needed since this is the scaling factor.

12. The process is repeated for the next problem.

The auxiliary arithmetic unit performs fixed decimal point operations and therefore the decimal point must be considered separately.

Conclusions

Accuracy

Practically, an average accuracy of $\pm$ .001% to $\pm$ .01% full scale can be achieved. Since most of the errors are fixed, it is particularly desirable to operate as close to full scale as possible. The variable gain amplifier helps achieve this. Some numerical methods have been considered for the purpose of improving the accuracy. In general, their shortcomings outweigh the increase of accuracy they may offer.

When all is said and done, digital/analog arithmetic will not be useful where accuracy is at a premium. However, there are some problems in which its accuracy may be sufficient. In some problems the accuracy of the information is limited, so that extreme accuracy of computation is unwarranted. Two examples of this occur in industrial process control and circuits analysis. In the former the input information is in analog form and in the latter, the most precise components are often 5% resistors.

Speed

In digital/analog arithmetic most operations take the same time, for example, the solution of:

$$y = a + b \tag{16}$$

would take the same time as:

$$y = a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} \ldots + a_{i8} \cdot b_{8j} \tag{17}$$

about 100 usec.

The average speed of computation for the single addition in the first example is 100 usec., whereas the average speed of the 8 multiplications and 7 additions in the second problem is 6.8 usec. per single computation. The latter speed is fast even when compared to high-speed digital arithmetic.

Another aspect of speed to consider is the number of times memory must be used in the course of computation, since in high-speed computation the memory may be the limiting speed factor. Therefore, a less conspicuous advantage of digital/analog arithmetic is that it requires fewer trips to memory for similar computation as compared to digital arithmetic.

A third important factor is that during the time the digital/analog arithmetic unit is computing, it can operate completely independant of the rest of the computer. This allows the computer to perform its necessary functions in parallel with the digital/analog arithmetic unit for most of its cycle. Also, because there is a long lapse of time between successive operations, a high percentage of this time can be utilized.

Acknowledgements

The author wishes to thank Mr. W. Brandenberg for his help in the writing of the report and Mr. D. A. Bourne for his encouragement in this work. He is also indebted to Mr. D. J. Grenier and Mr. Secundo Decceco for their work on the A/D Converter and Tester.

References

1. Skramstad, H. K., "A Combined Analog-Digital Differential Analyzer", Proc. of EJCC, 1959.

2. McLeod, J. H. and Leger, R.M., "Combined Analog and Digital Systems", Instruments and Automation, Vol. 30, pp 1126.

3. "Applications of AD-DA Verter Systems in Combined Analog Digital Computer Operation", Pacific General Meeting of the AIEE, June '56, Paper #56-842.

4. Leger, R. M. and Greenstein, J.L., "Simulate Digitally or by Combining Analog and Digital Computing Facilities", Control Engineering, Sept. 1956.

5. Skramstad, H. K., Ernst, Nigro, J.P., "An Analog-Digital Simulator for the Design and Improvement of Man-Machine Systems", EJCC.

6. Blanyer, C. G. and Mori, H., "Analog, Digital and Combined Analog-Digital Computers for Real Time Simulatinn".

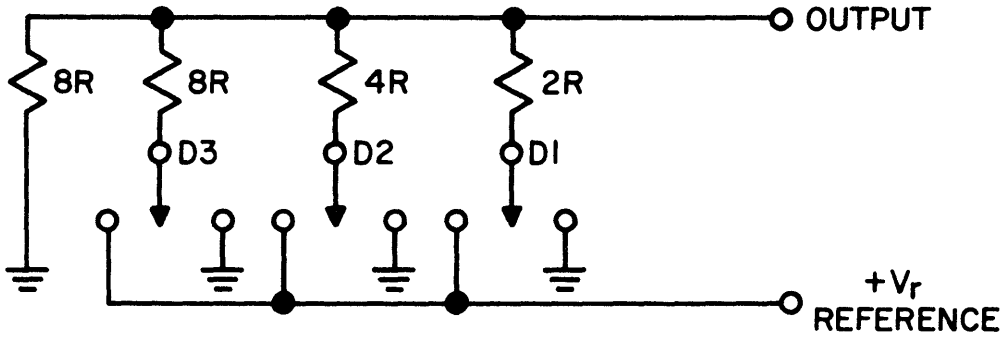


FIGURE 1-THREE BIT D/A CONVERTER

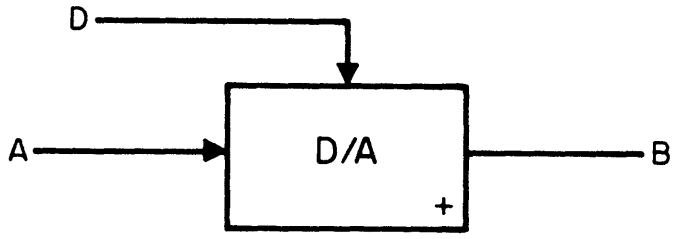


FIGURE 2-DIGITAL TO ANALOG CONVERTER

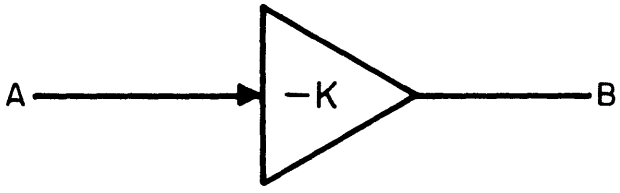


FIGURE 3-SUMMING AMPLIFIER

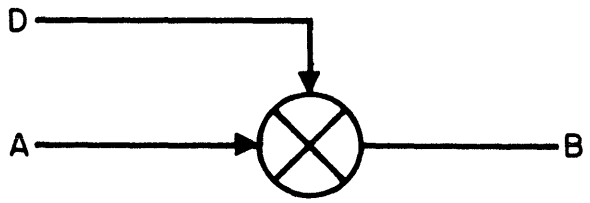


FIGURE 4-MULTIPLEXER

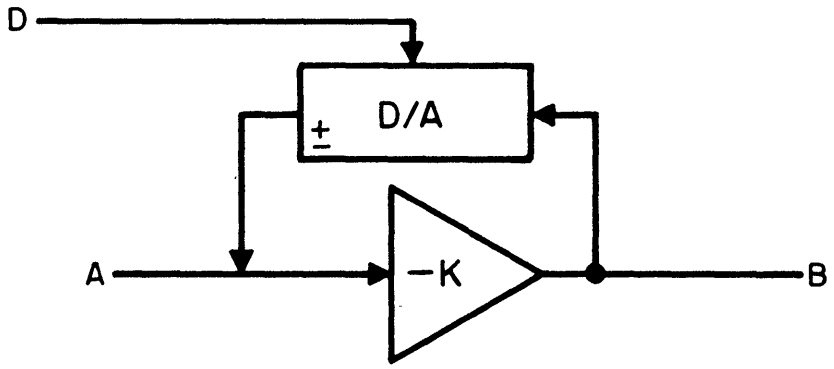


FIGURE 5-VARIABLE GAIN AMPLIFIER

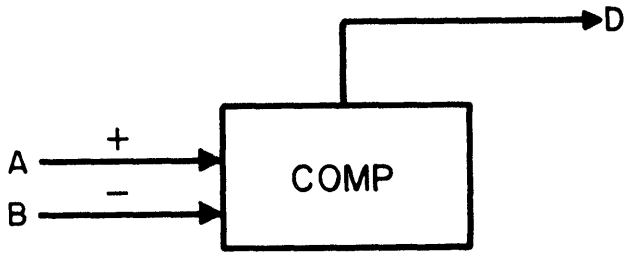


FIGURE 6-COMPARATOR

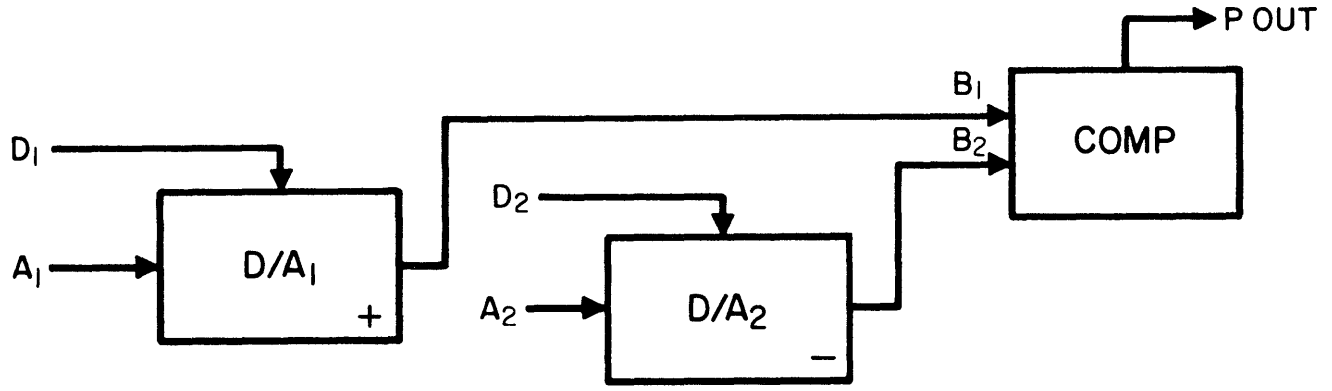


FIGURE 7–TWO D/A CONVERTERS FEEDING A COMPARATOR

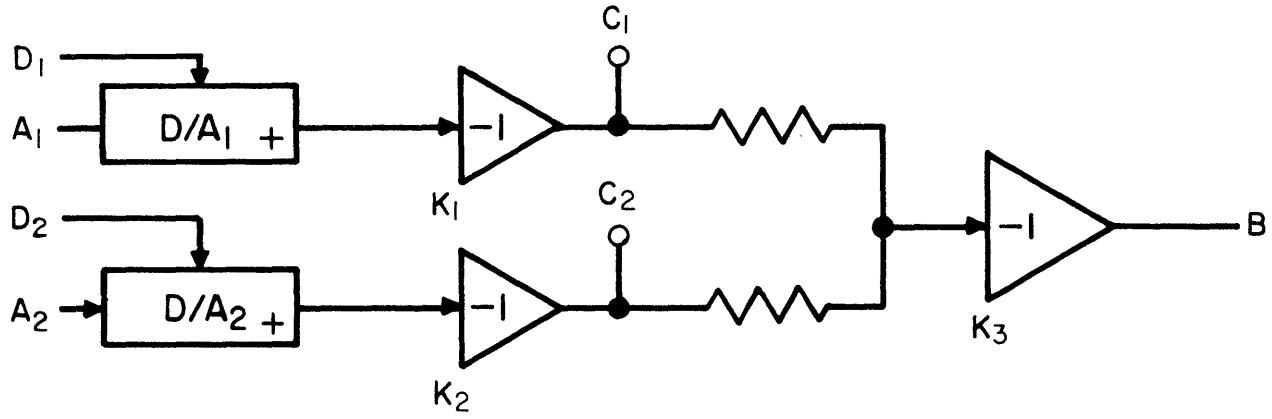


FIGURE 8 — LOGIC FOR SUMMING TWO PRODUCTS

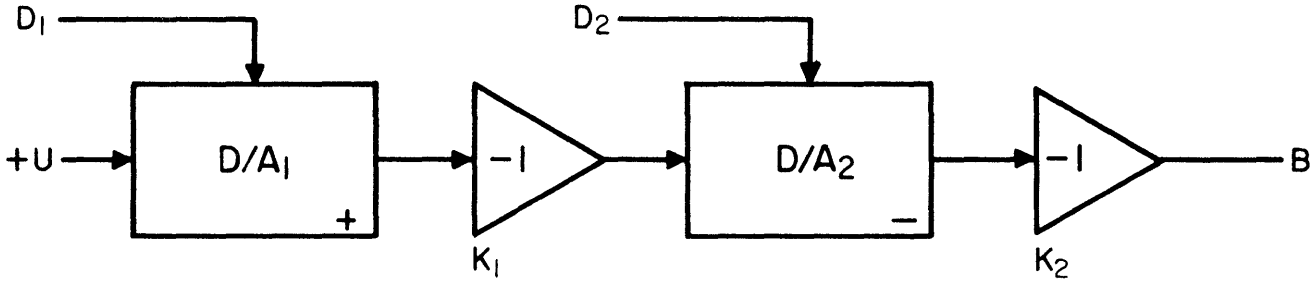


FIGURE 9–MULTIPLICATION OF TWO DIGITAL QUANTITIES

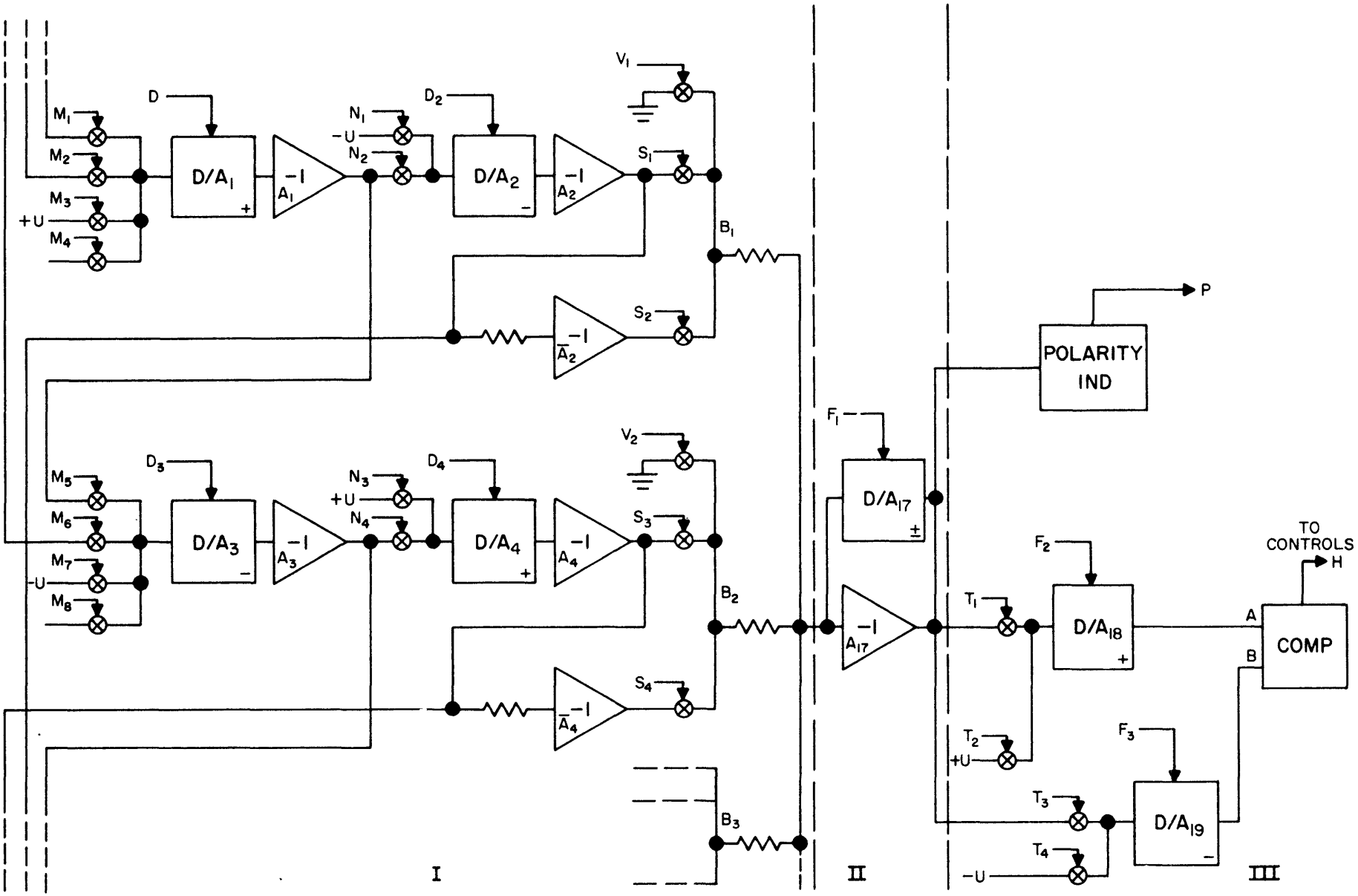


FIGURE 10 – DIGITAL/ANALOG ARITHMETIC UNIT

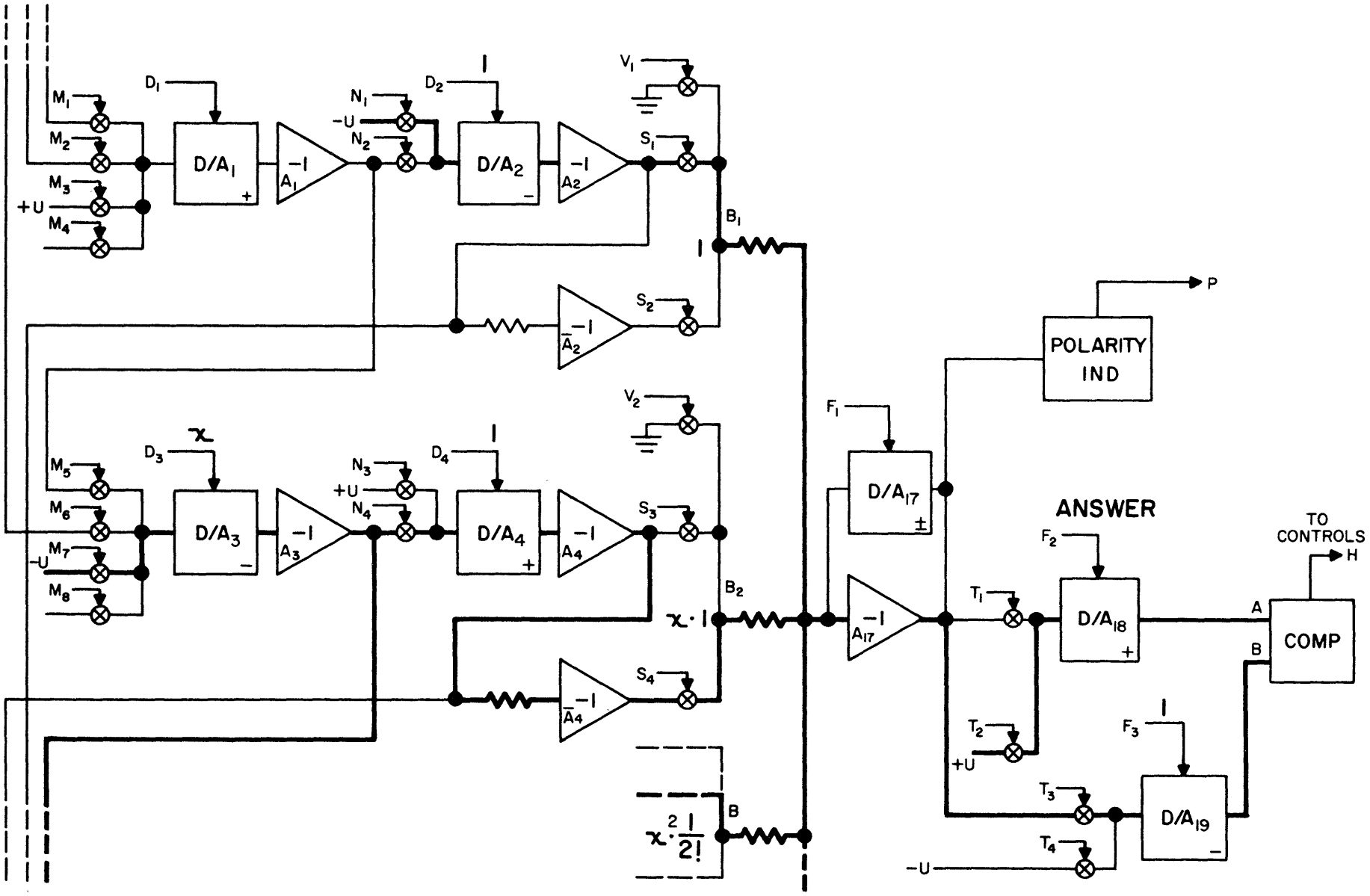


FIGURE 11 – EXAMPLE I

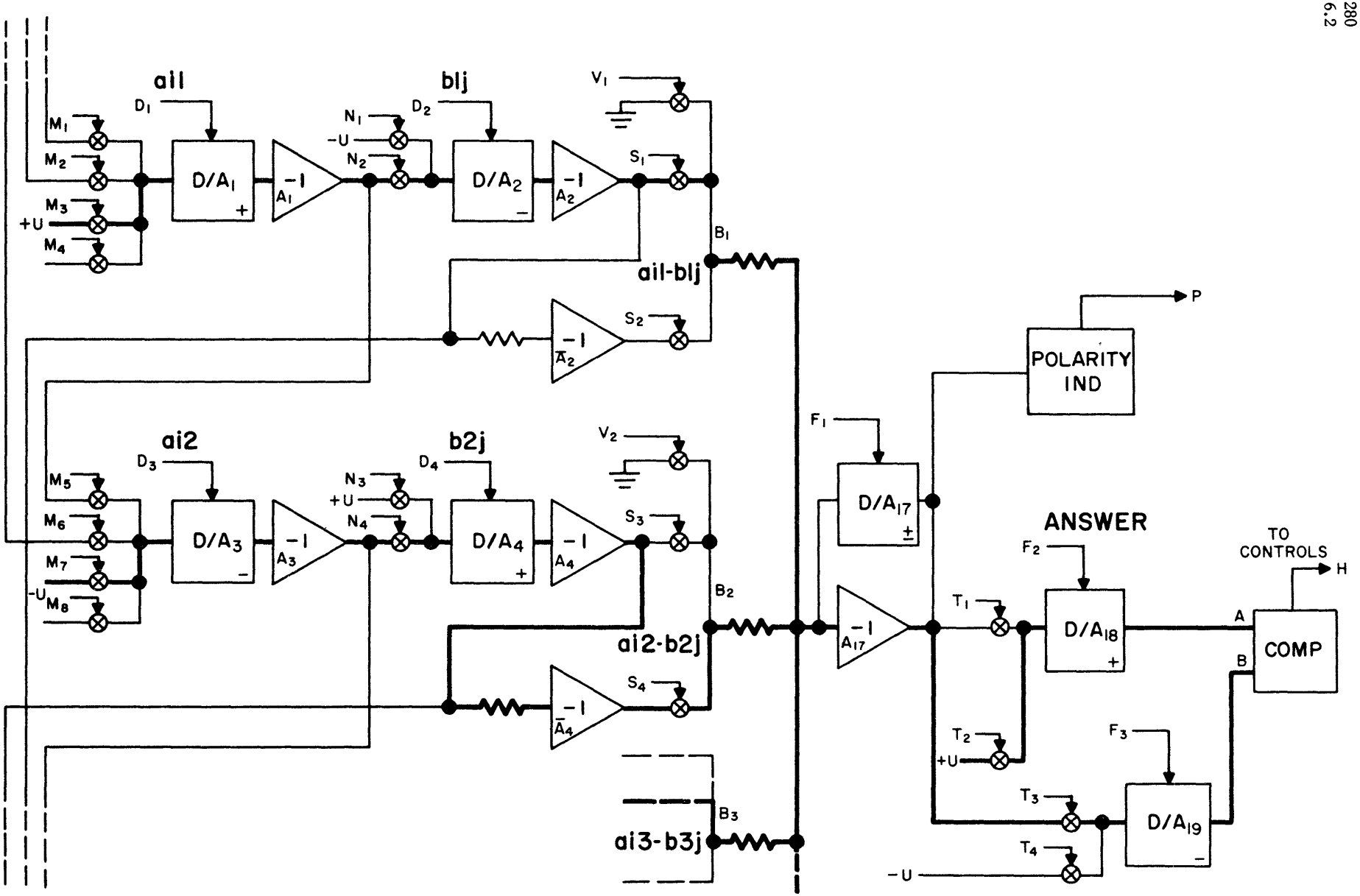


FIGURE 12- EXAMPLE II

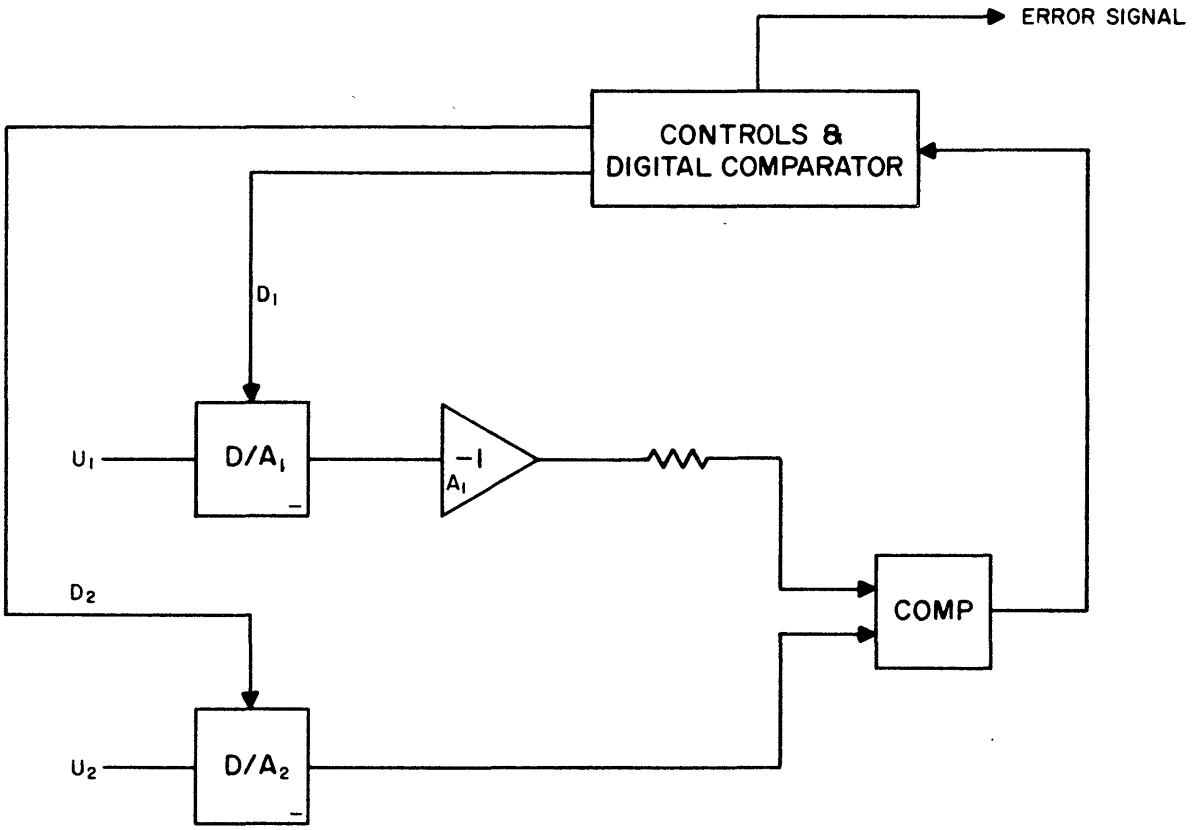


FIGURE 13–SCALE MODEL OF D/A ARITHMETIC UNIT
(CONVERTER–TESTER COMBINATION)

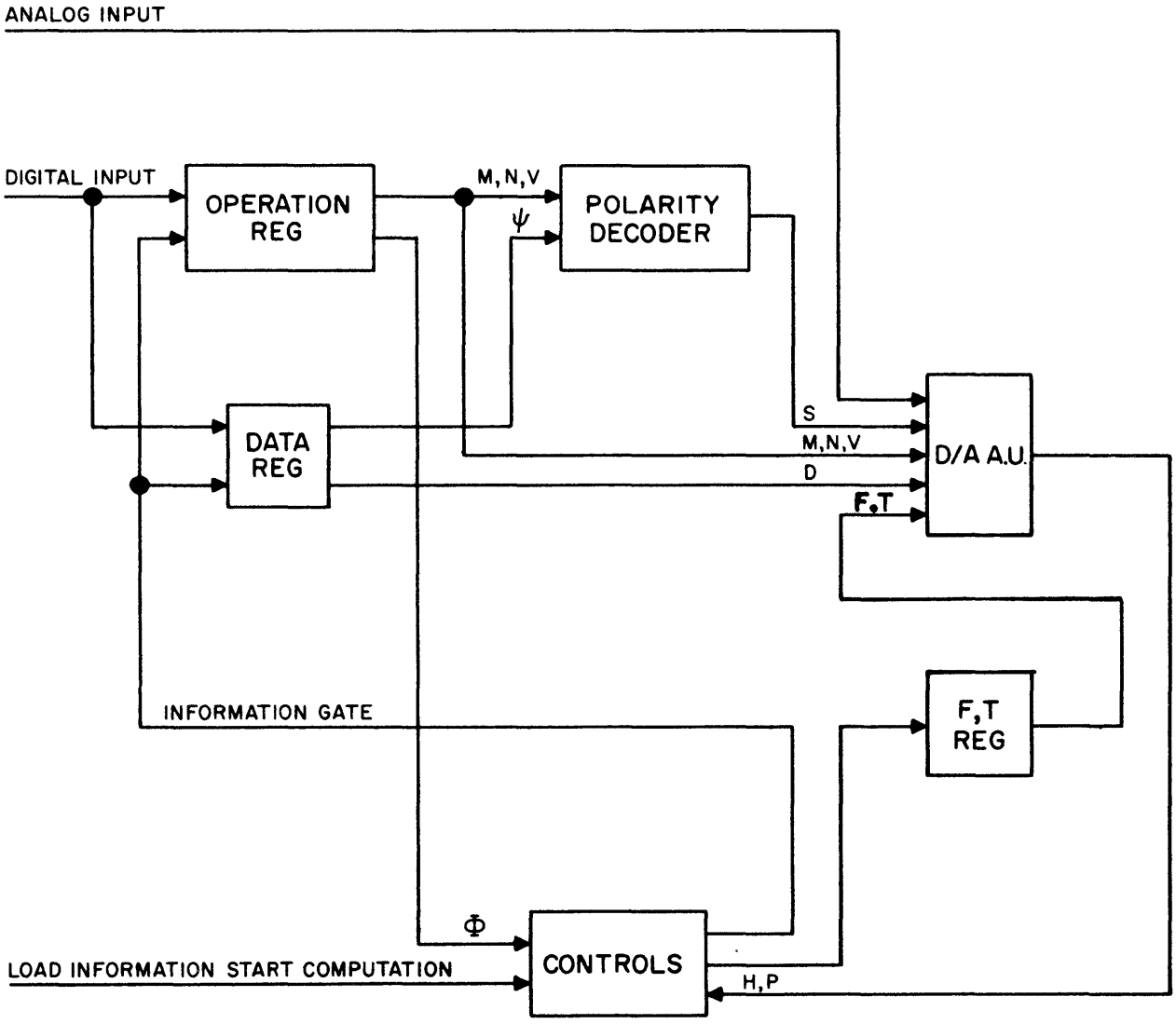


FIGURE 14 AUXILIARY ARITHMETIC UNIT

PB 250

A HIGH-SPEED SERIAL GENERAL PURPOSE COMPUTER USING MAGNETOSTRICTIVE DELAY LINE STORAGE

By
Robert Mark Beck
Packard Bell Computer Corporation
Los Angeles, California

Summary

This paper presents the design objective of a general purpose computer which is intended to serve as a component in special purpose systems.

Some of the design considerations applied toward meeting these objectives are also presented. The significance of the use of magnetostrictive delay memories in a low-cost computer as well as the approach used to minimize the active elements in a flexible computer input/output system are discussed.

Introduction

For the past three years, Packard Bell Computer Corporation has designed and developed special purpose data gathering and data handling systems.

A typical system function is to translate several channels of analog information into a prescribed digital format and record the digital information on magnetic tape. In these systems, an electronic multiplexer selects the proper channel of input voltage and holds it for digitalization by an analog-to-digital converter. A special digital control unit then edits and arranges the digital information into the required format. The digital information is then recorded on magnetic tape.

Another typical system function is to generate control signals as a function of several analog inputs. The control signals thus generated are used to select, operate, or control other units. A system of this type may become very extensive, with many registers required for such functions as accumulation of data, multiplication, memory, shifting, or intermediate storage of data. Therefore, even a low-performance system of this type may require over 200 flip-flops and 500 words of memory.

Upon investigation of the special purpose system approaches described above, the following major disadvantages were determined:

a. It is difficult to meet short schedules. Large amounts of expensive design and development time are required since each system must be built for its special function.

b. The special purpose system is not flexible; therefore, when it becomes obsolete, only a minimum of the components can be salvaged.

c. Because of the relatively short life of special purpose systems, maintenance knowledge and records can not be effectively accumulated.

Examination of the above facts compared with general purpose computer characteristics resulted in the following conclusion: A special purpose system could have a major portion of its disadvantages eliminated by making the heart of the system a general purpose computer, especially designed with a flexible input/output which would allow it to operate with other equipment in the system. Of course, the crux of this idea is to have the majority of the system engineering expressed in computer programming instead of special hardware. The major advantages to be derived from this method of implementing special purpose systems are as follows:

a. Design and development time are minimized since a large part of the system function is performed by the computer. Only the program of the system must be developed, and portions of it may be existing subroutines.

b. The system is flexible since changes are easily incorporated in the program to meet new or revised system requirements.

c. The computer continues to be useful even after the special purpose system becomes obsolete. The major components, including the computer, may be kept intact for use in future systems. The program is the only part of the system which becomes obsolete.

d. The reliability and ease of maintenance are increased because continual maintenance knowledge and records will be retained for the

computer, which is a major portion of the system. Check routines may be incorporated to ensure that the computer is operating correctly.

Design Specifications

A computer to be used as a systems component, as set forth in the introduction, requires stringent design specifications.

The computer must be very fast to perform sequentially the operations of a special purpose digital system. Furthermore, a low component count is necessary to make the computer cost competitive with the special equipment it replaces.

The communication required with other units in a system dictates a flexible input/output. Many input/output channels with many codes and formats must be available.

To be economical in all cases, the memory capacity must be expandable so that the memory capacity can suit the size of the job. Furthermore, programming should be made as easy as possible in order to keep the programming time and expense less than the engineering which it replaces. This suggests a complete command list using single address operation. Finally, if the computer is to be a component in a system, it should be compact and suitable for rack mounting.

Analysis of the above requirements led to the design goals as follows:

1. High-Speed Operation - -
   50,000 commands per second
2. Minimization of Components - -
   a. 30 Flip-Flops
   b. 300 Transistors
3. Flexible Input/Output - -
   a. Basic:
   Flexowriter with Reader and Punch
   b. Other Equipment:
   Magnetic Tape Handlers
   High-Speed Paper Tape Punches
   High-Speed Paper Tape Readers
   Analog Voltage Multiplexers
   Digitizers
   Digital-to-Analog Converter
   Card Punches and Readers
   External Core Memories
4. Expandable Memory - -
   a. Basic - - 2,000 words
   b. Expanded - - 16,000 words
5. Easy Programming - -
   a. Complete Set of Instructions
   b. Single Address Operation
   c. Double Precision Commands
6. Word Size - - 21 Bits plus Sign
7. Compact Construction - -
   Suitable for Rack Mounting

Computer Description

Using these design goals, the first design conferences were started in November 1959. A 10-man engineering team took part in these design conferences. This team worked simultaneously on the logic, circuitry, and mechanical design of the proposed computer.

In August 1960, a prototype of the computer was placed in operation. Circuitry reliability testing and program checking have been continuously performed on this computer since it was placed in operation.

During the prototype checkout, ten production computers were built. The first production computer was delivered in October 1960.

Memory Elements

The fundamental design decision was to chose the type of memory element. Initially, a magnetic drum with a 1 mc bit rate and speed of 400 cps, and a low-speed core memory with a cycle time of 10 microseconds were considered.

Our attention then turned to magnetostrictive delay lines, as a result of articles published by the Ferranti Company and Arma Corporation on the use of magnetostrictive delay line memories in the Pegasus computer and the Titan Missile test computer, respectively. An expandable memory could be made very conveniently by packaging each memory line as a plug-in module. Based on a temperature coefficient of 0.5 PPM/^{o}C, 6,144 bits could be stored in each memory line register. This would realize 256 words of 24 bits each per memory line. Furthermore, using a 2 mc NRZ writing process, the desired goal for a high-speed serial operation would be practical. Hence, based on these advantages, magnetostrictive

delay lines were chosen for the memory of the PB 250 general purpose computer.

Figure 1 shows a complete memory line module. This circuit card includes line address selection circuits in addition to the line and the reading and writing circuits. The writing circuit consists of a flip-flop which is DC coupled to the magnetostrictive line's input transducer. The reading transducer presents a 2 mv peak-to-peak differential signal. This signal is amplified, then reshaped by a Schmitt circuit. The output of the Schmitt circuit is gated into the read flip-flop. The waveforms as they appear at various points in the delay line register are shown in Figure 3.

Figure 2 shows a one-word delay line module which is used in arithmetic portions of the computer. This circuit circuit card is smaller because it does not contain line address selection circuitry.

Computer Circuitry

The characteristics of the basic circuits used in the PB 250 are outlined below:

1. Gating - -
   a. Voltage Levels:
      0 volts (" false" level)
      - 8 volts (" true" level)
   b. Structure:
      Two-level AND gates feeding OR gates.
2. Clock Waveform - -
   Single phase, 2 mc square wave.
3. Flip-Flops - -
   a. Construction:
      Two Transistors (no clamping)
   b. Input Coupling:
      Capacitor - Diode
4. Emitter Followers - -
   One-Transistor
5. Inverters - -
   One-Transistor

Where delays would become too great, transistor OR gates are formed by connecting the emitters of several emitter followers. Figure 4 shows the basic circuit configurations (specialized circuits such as solenoid drivers and clock generators are omitted). All circuits were designed to achieve reliable performance with a minimum of semiconductor elements. These circuits were derived from 3 mc circuits used in Packard Bell Computer Corporation's TRICE Computer (a digital differential analyzer system).

Computer Organization

The computer word is made up of 24 bits. Two of the bits, the first (a guard bit) and the 24th (an odd parity bit), are not available to the programming. Numbers in the machine are expressed in binary with 21 bits plus sign. The command format is indicated in Figure 5. This block diagram also shows all the basic elements of the PB 250. The heavy-lined rectangles locate the five one-word magnetostrictive delay line registers. The computer's 32 flip-flops (excluding reading and writing flip-flops for magnetostrictive delay line registers) are summarized in Table 1.

Arithmetic Operations

The computer's arithmetic operations are based on the three one-word registers: A, B, and C.

For double precision operations, the A and B registers are arranged to handle double precision numbers with the sign and 21 most significant bits of the number in A and the 22 least significant bits in B. The basic arithmetic operations are outlined below:

1. Addition and Subtraction - -
   Addend or Minuend ⟶ A or A and B
   Augend or Subtrahend ⟶ Memory
   Results ⟶ A or A and B
2. Left and Right Shifting - -
   A Register and B Register shifted as a pair, C Register is incremented or decremented to allow for efficient floating point subroutines.
3. Multiplication - -
   Multiplier ⟶ B Register
   Multiplicand ⟶ C Register
   Product ⟶ A and B Registers
4. Division - -
   Numerator ⟶ A or A and B
   Denominator ⟶ C Register
   Quotient ⟶ B Register

5. Square Root - -

Argument ——————→ A or A and B

Square Root of
Argument ——————→ B Register

Each of the short registers recirculates once per word time. Therefore, multiply, divide, and square root commands produce one new bit in the answers for each execution word time. The shift operations require one word time per shift. These commands are executed for a programmed duration, for improved optimum programming.

Timing

Two methods of programming exist for the PB 250. The normal computer program consists of reading successive commands in sequence from a command line. This method offers optimum storage usage. Every 3-millisecond cycle of a long memory line, a new command is read and executed when the operand in the command appears. Hence, the computer waits a major portion of the time. An operation rate of 333 commands per second is realizable using this method of programming.

To provide high-speed operation, an optimum time programming technique is available. Any command which contains an optimum code bit is called an optimized command. The command immediately available at the completion of the execution of the optimized command is read. Hence, the computer can be reading or executing a command at all times. Using optimum time programming, a computation rate of 40,000 commands per second is obtainable.

The execution time for addition or subtraction is 12 microseconds for single precision numbers and 24 microseconds for double precision numbers. Execution of left shifting, right shifting, multiplication, division, and square root commands requires 12 microseconds per word time of execution. As previously mentioned, the length of execution of these commands is programmed within the structure of the command. For example, a full-length, 22-bit plus sign multiplication requires 276 microseconds. Hence, with optimum time programming, the PB 250 can perform 2,800 full-length multiplications per second.

Input/Output Operation

The bulk of the input/output capability of the PB 250 is designed into the command structure. The computer handles characters in any code configuration (up to 8 bits per character) for input/output operation.

Each output character is delivered by an output command which has a programmed duration. This programmed duration is controlled by using the C Register as a counter to provide output character signals ranging from 12 microseconds to 24 seconds.

Each 8-bit input character is entered into a small input buffer contained in the computer. These 8-bit input characters are assembled into words by computer programming. The same computer circuits are used to handle character rates of 10 cps to 2,000 cps in an efficient manner.

Computer handling of each character by programmed operation removes any restrictions on codes or formats and also minimizes the amount of hardware applied to input/output operation in the computer. High data transfer rates may be obtained by using an external buffer. For control applications, the computer's complement of links with external equipment is completed by its ability to sense the many input lines and generate many " string pulling" output signals.

An external 2 mc shift register is provided as auxiliary equipment for large, quick bursts of input or output data. This is especially useful when dealing with analog-to-digital and digital-to-analog converters, card readers and punches, line printers, etc.

The computer may be programmed to sense many parallel input lines. This capability is enhanced by a command which allows program branching based on any of 30 input lines. The converse is available for output operation; that is, output pulse signaling to any one of 30 output lines.

Finally, to provide a maximum of flexibility for input and output, a special provision is incorporated for synchronizing and interconnecting two or more PB 250's. With this arrangement, one computer may serve as the central computer, while other computers may be used for input and output functions.

Packaging

Figures 6, 7, and 8 show the computer as a rack-mounted component. The PB 250 requires 33 1/4 inches of vertical rack space. The Flexowriter requires an additional 17 1/2 inches of rack space.

The computer frame can accommodate up to 15 memory lines (approximately 4,000 words). Additional memory lines may be mounted in another chassis. Any 256-word memory line

may be replaced by a shorter line, such as a 16-word line, for increased fast access memory.

The total component count includes only 375 transistors and 2,300 diodes. There are 120 plug-in circuit modules. The open-book case style, as shown in Figure 7, allows easy access to all the socket wiring of these modules.

For program tracing, the control panel at the left side of the front panel indicates the status of the static flip-flops. The marginal test switches located on the control panel are used for varying the clock period. Further marginal checking may be performed by adjusting the two main power supply voltages, + 6 volts and - 12 volts.

Performance Review

Although the PB 250 Computer was designed to be a useful system component, it has also proved to be a powerful general purpose computer. With an automatic optimum programming assembly routine and a library of routines and subroutines, the PB 250 provides the greatest advance in low cost computers since magnetic drums were first used.

Efficiency of the computer is greatly increased by the use of the solid-state devices such as transistors, diodes, and magnetostrictive delay lines. The efficiency of these components is made possible by using a 2 mc operating frequency (about 10 times the frequency of drum computers). This frequency also allows more efficient logic coupling between the input/output equipment and the computer. Hence, the burden of input and output operations is placed on programmed subroutines, thereby reducing input/output hardware, and thus compensating for the expense of high-frequency components.

The PB 250 approaches or exceeds the design objectives that were set for it. A large number of special system programs are now being prepared with very satisfactory performance indicated. These systems include tracking radar antenna control, power plant control, atomic reactor data logging, etc.

The complete PB 250 command list is presented in Table 2. The commands in the top half of the list are the basic single address operations, whereas the commands in the lower half of the list have a programmed execution duration which starts in the word sector after the command is read and ends at the address specified by the command.

Acknowledgements

The author wishes to express his appreciation to Dr. Stanley Frankel for his valuable advice and consulting assistance in the design of the PB 250 logic. Congratulations are also extended to Mr. Smil Ruhman for his outstanding work on all aspects of the computer's circuit design and to Messrs. Jack Mitchell and Donald Cooper for their efforts in the management and coordination of the overall engineering project.

Table 1.

Flip-Flops	Function
F1, F2, F3, F4, F5	Pulse Time Counter (P1 - P24)
Ec, Rc	Phase Control - - $\overline{Ec}\ \overline{Rc}$: Wait to Read Command $\overline{Ec}$ Rc : Read Command Ec Rc : Wait to Execute Command Ec $\overline{Rc}$: Execute Command
Is	Comparison Detector - - Compares Sector Counter and Instruction Register
Oc, O6, O5, O4, O3, O2, O1	Operation Code Register
L5, L4, L3, L2, L1	Operand Line Register
K3, K2, K1	Command Line Register
Sc	Carry for Sector Counter
Ca	Carry for Arithmetic Unit Adder
Of	Overflow
Pc	Parity Check
Ae, Be, Ce	Arithmetic Unit Register Shift Flip-Flops
Rf, Tf	Reader and Typewriter Controls

Table 2.

OP		Command	OP		Command
00	HLT	Halt	40	EBP	Extend bit pattern of A
01	IAC	Interchange A and C	41	GTB	Convert A from Gray to Binary
02	IBC	Interchange B and C	42	AMC	AND of M & C
03	- -	- -	43	CLB	Clear B
04	LDC	Load C	44	CLC	Clear C
05	LDA	Load A	45	CLA	Clear A
06	LDB	Load B	46	AOC	AND-OR Combined
07	LDP	Load AB double precision	47	EXF	Extract Field
10	STC	Store C	50	DIU	Disconnect Input Unit
11	STA	Store A	51	RTK	Read Typewriter Keyboard
12	STB	Store B	52	RPT	Read Paper Tape
13	STD	Store AB double precision	53	RFU	Read Fast Unit
14	ADD	Add	54	- -	- -
15	SUB	Subtract	55	LAI	Load A from Input
16	DPA	Add double precision	56	CAM	Compare A and M
17	DPS	Subtract double precision	57	CIB	Clear Input Buffer
20	NAD	Normalize AB	60	WOC	Write Output Character
21	LSD	Left Shift AB	61		
22	RSI	Right Shift AB	62		
23	SAI	Scale AB	63		
24	NOP	No Operation	64		
25	IAM	Interchange A & M	65		
26	MLX	Move Memory Line to Line 7	66		
27	- -	- -	67	WOC	Write Output Character
30	SQR	Square Root of AB to B	70	PTU	Pulse Specified Unit
31	DIV	Quotient of AB÷C to B	71	MCL	Move Command Line
32	MUP	Product of B x C to AB	72	BSO	Block Serial Output
33	- -	- -	73	BSI	Block Serial Input
34	TCN	Transfer if C negative	74	- -	- -
35	TAN	Transfer if A negative	75	TOF	Transfer on Overflow
36	TBN	Transfer if B negative	76	- -	- -
37	TRU	Transfer unconditionally	77	TES	Transfer on External Signal

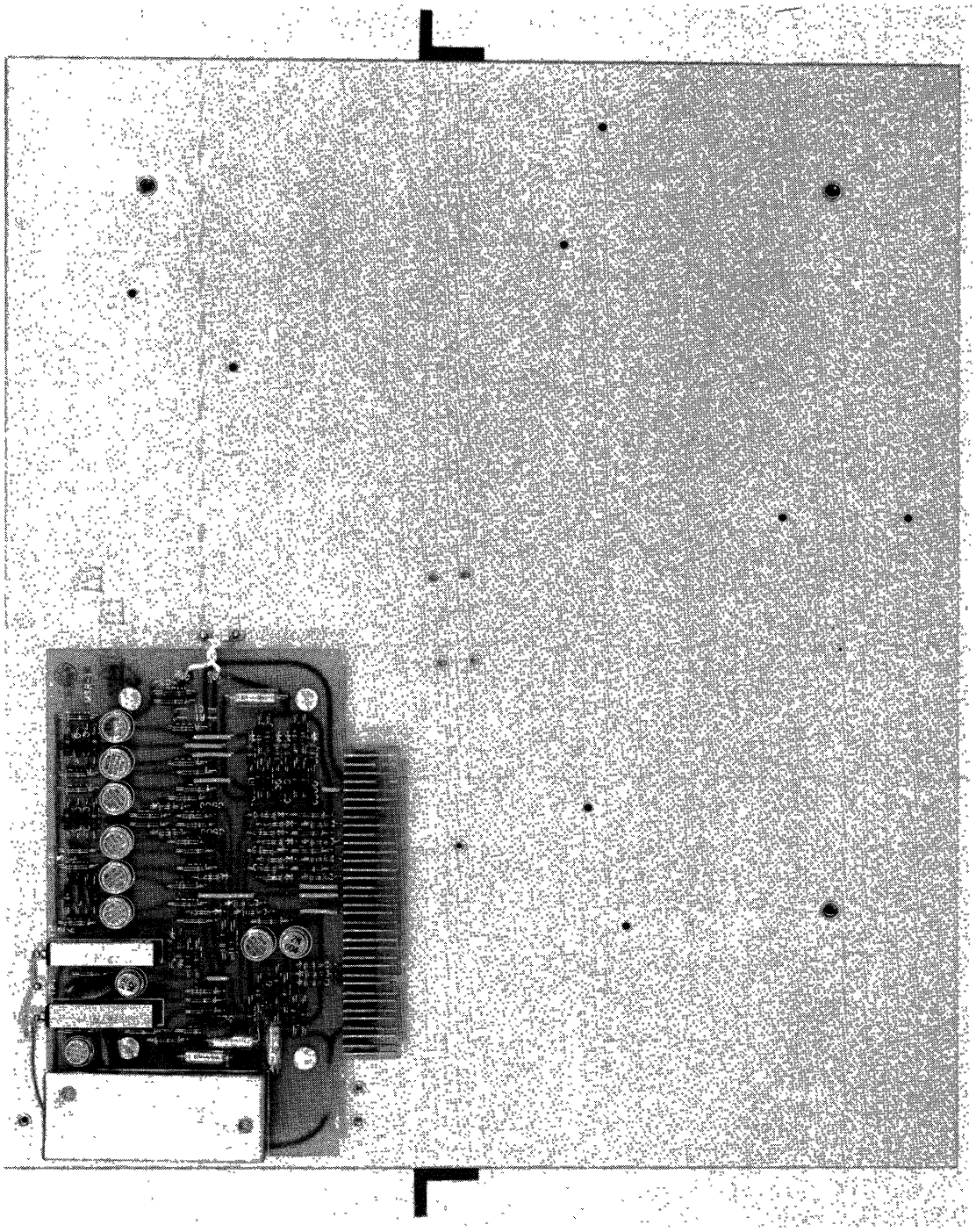


Fig. 1. Complete Memory Line Module

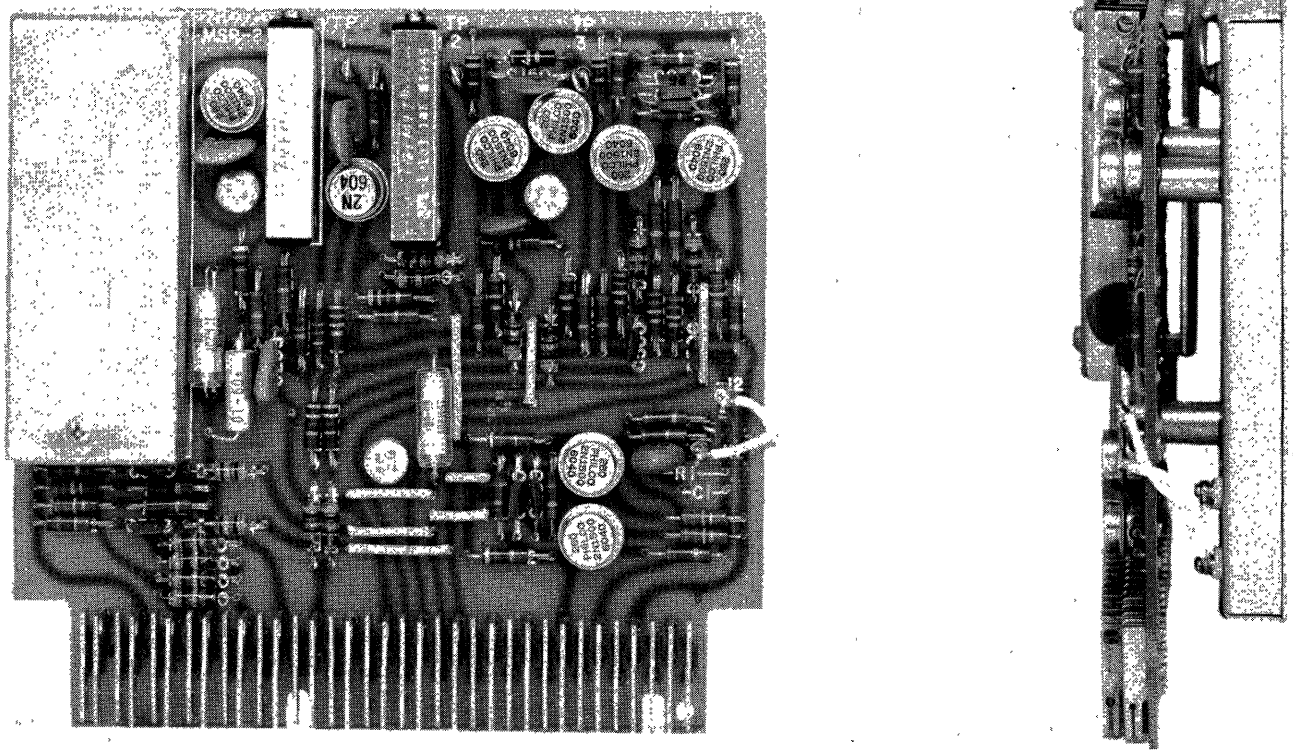
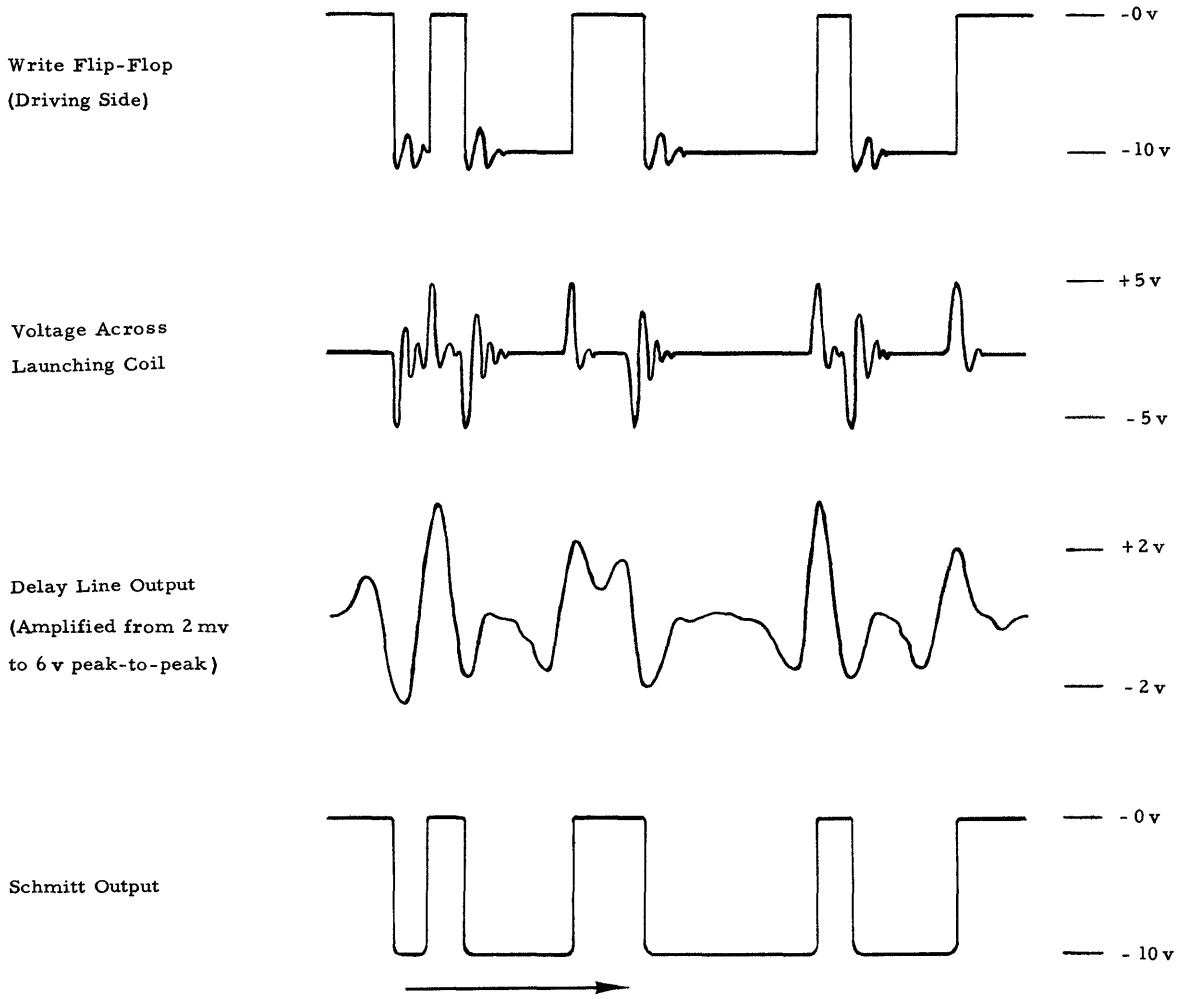


Fig. 2. One-Word Delay Line Module



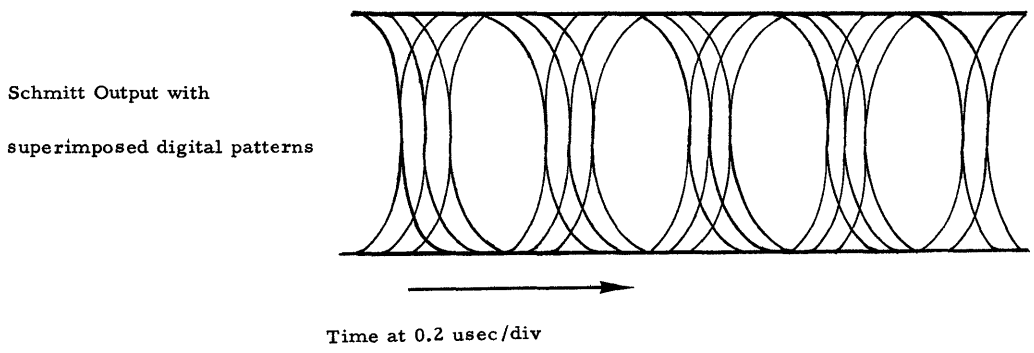


Fig. 3. Delay Line Waveforms

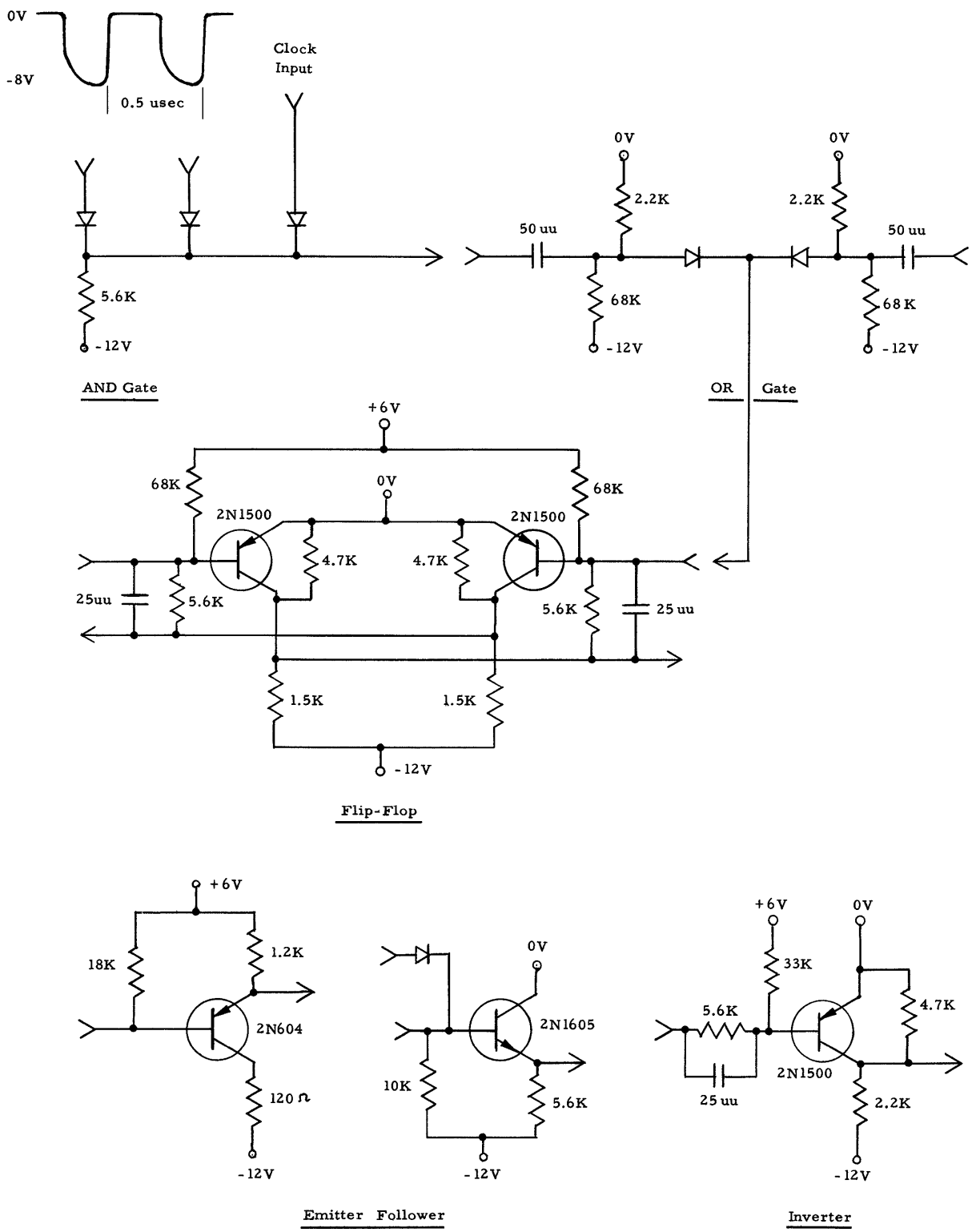


Fig. 4. PB250 Circuits

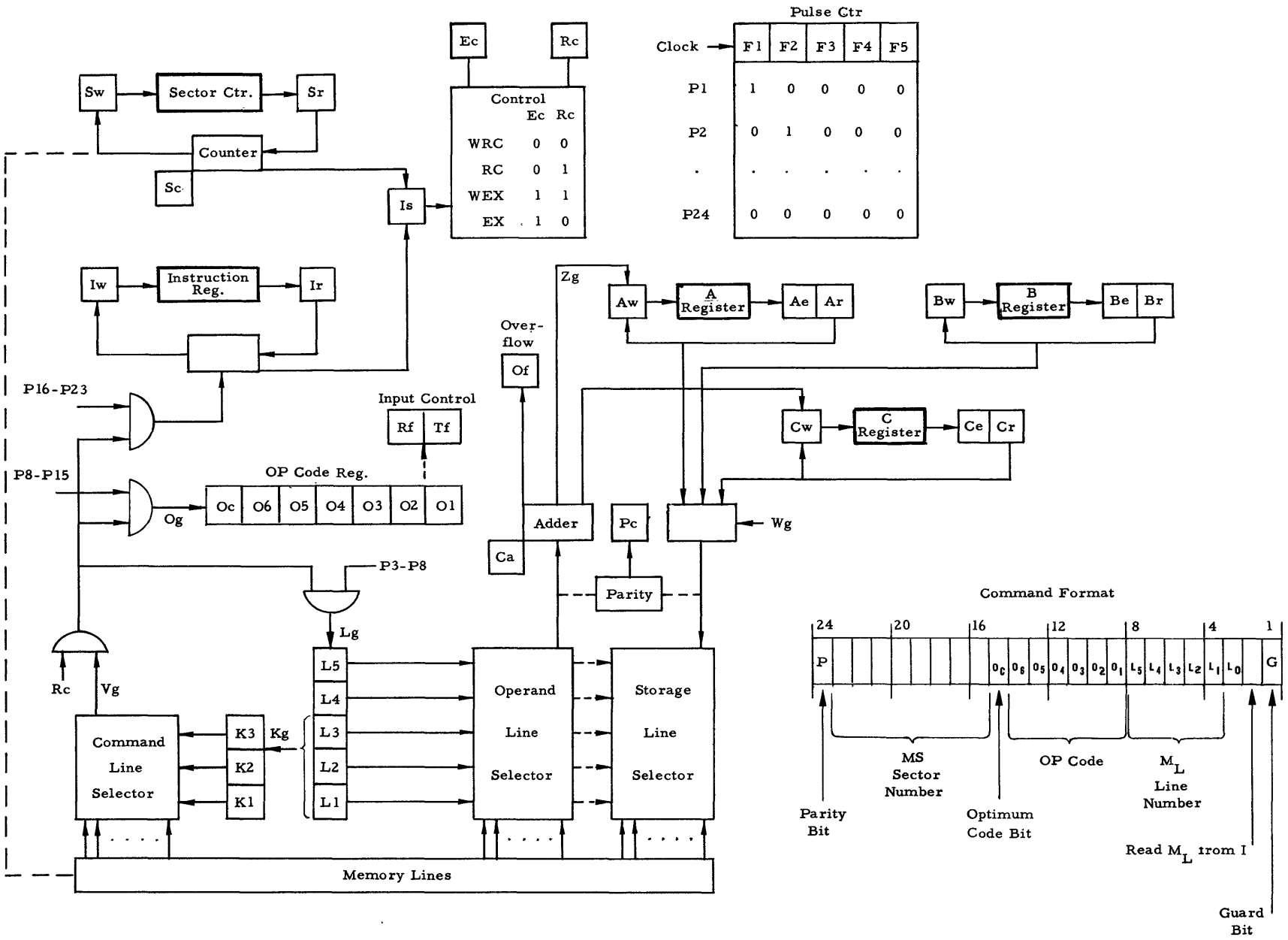


Fig. 5. PB250 Block Diagram

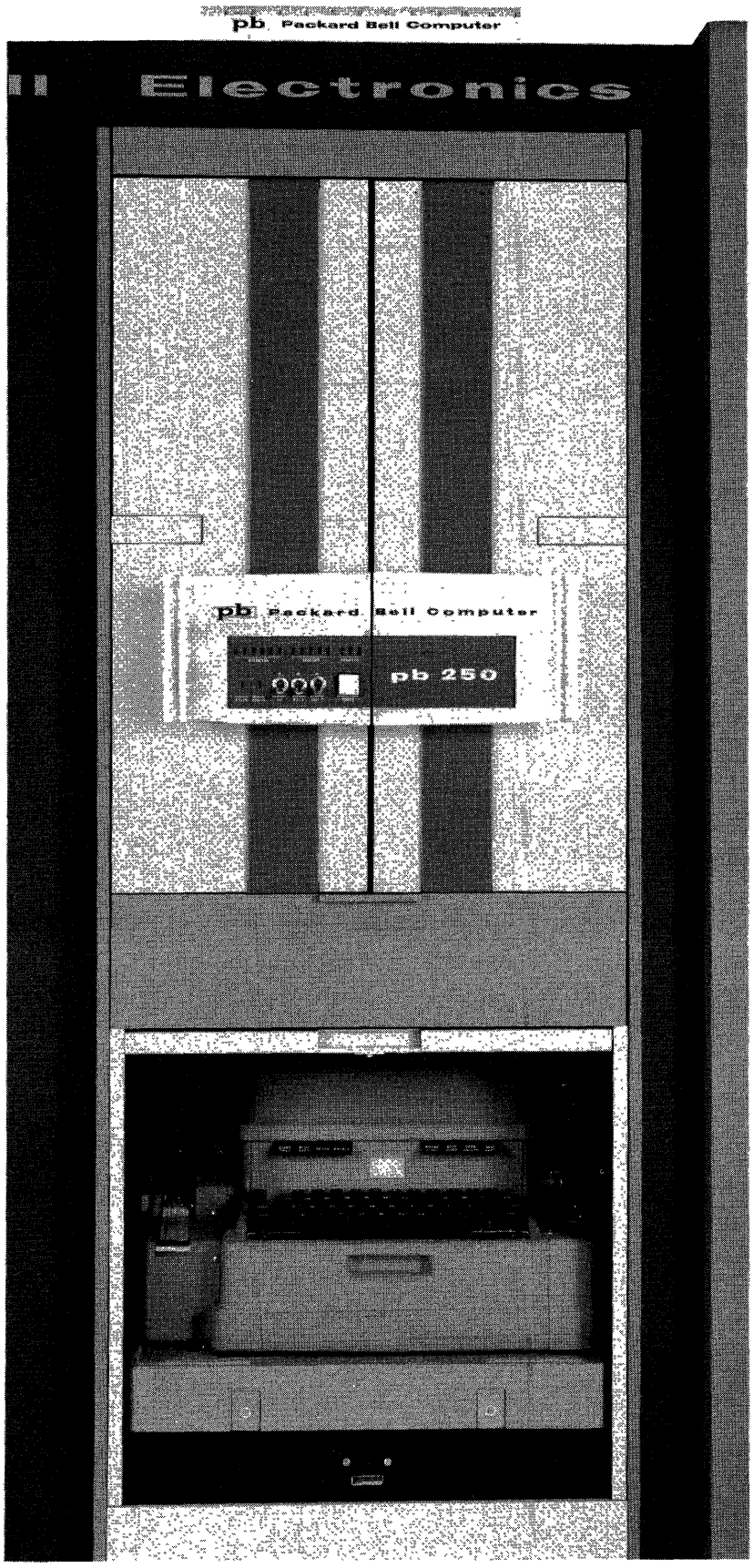


Fig. 6. PB250 Rack Mounted

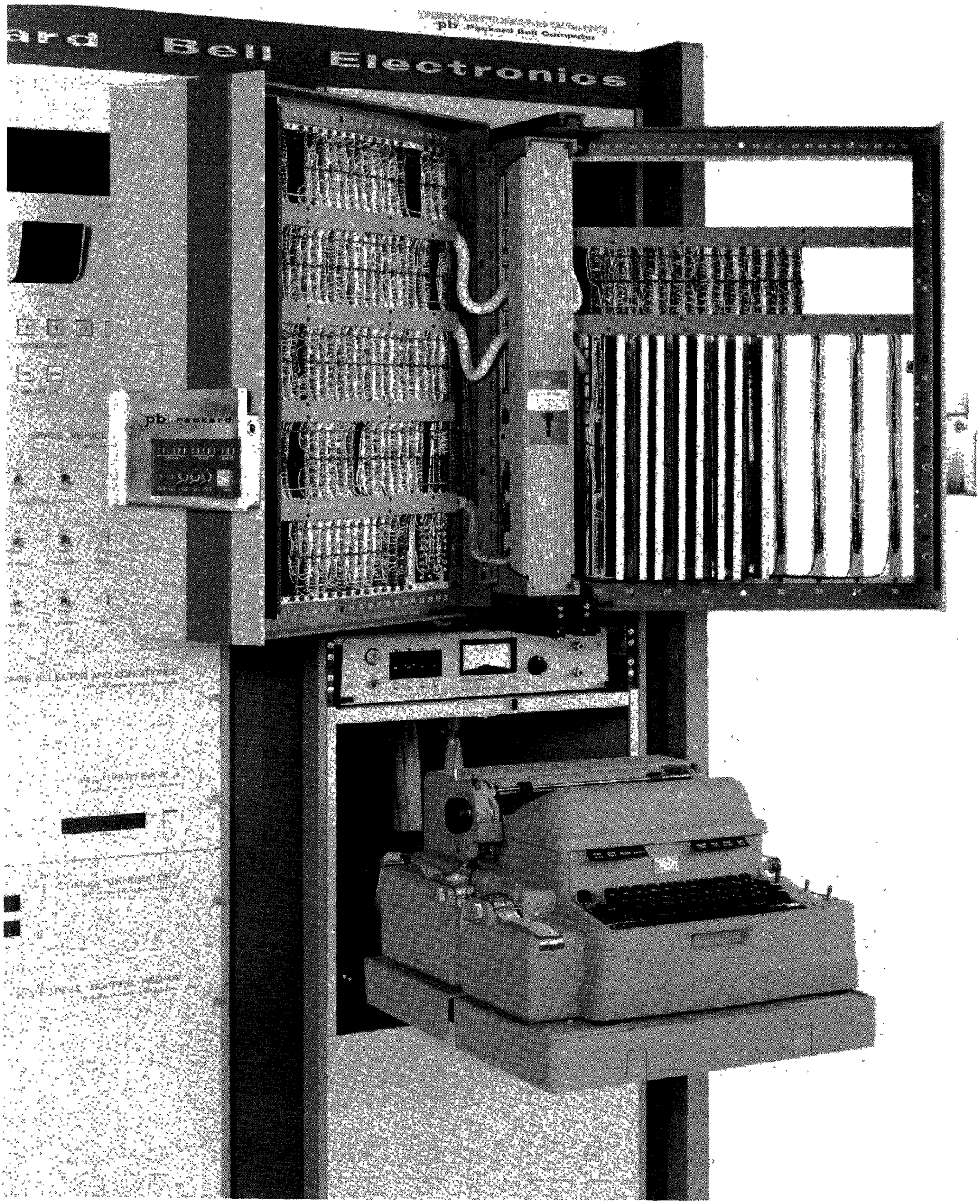


Fig. 7. PB250 With Case Opened

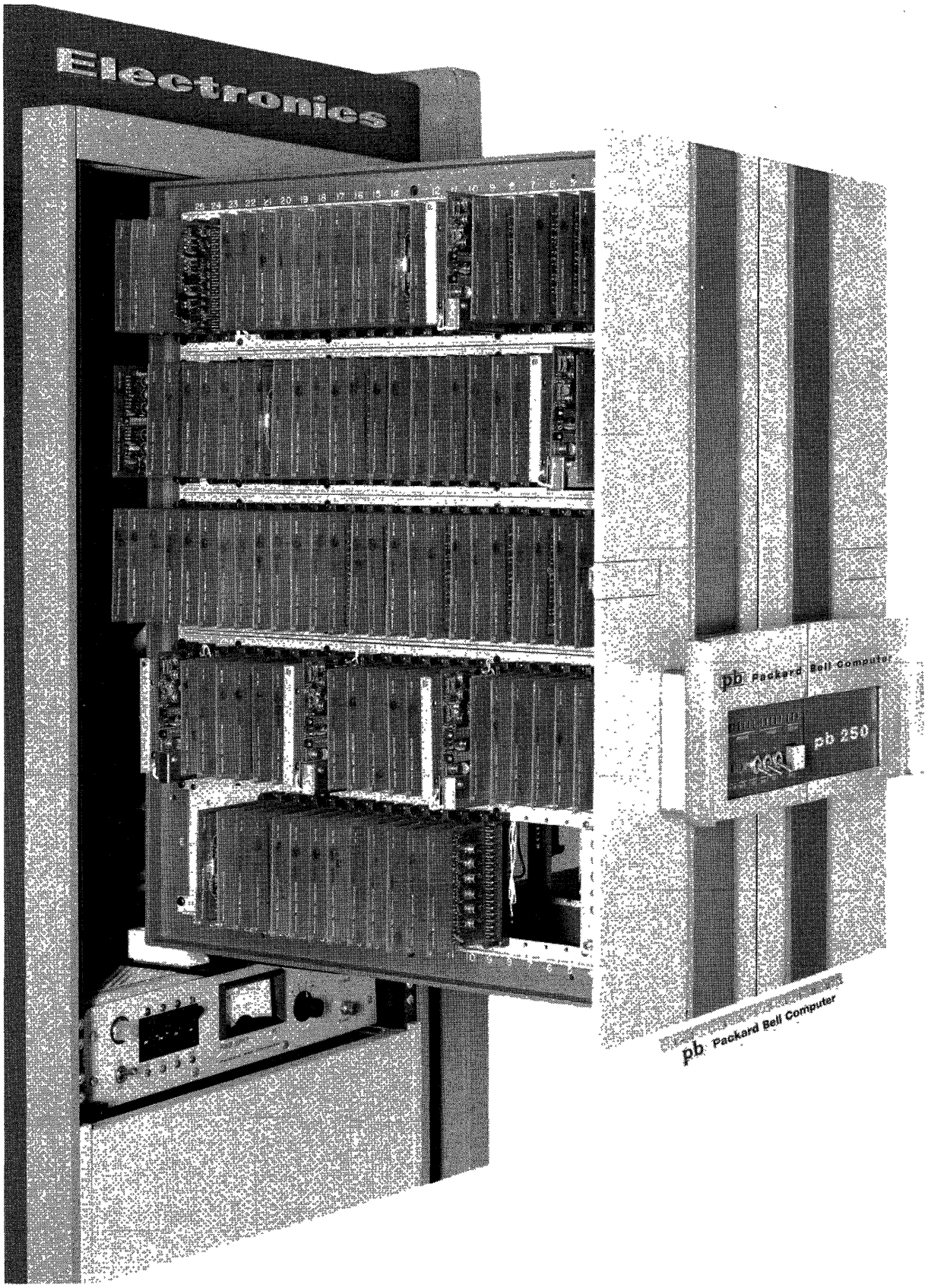


Fig. 8. PB250 With Case Pulled Out

THE INSTRUCTION UNIT OF THE STRETCH COMPUTER

R. T. Blosk
Product Development Laboratory, Data Systems Division
International Business Machines Corporation
Poughkeepsie, New York

Introduction

The Instruction Unit (I unit) was developed as the largest portion and the major control unit of the large-scale, high-performance Stretch computer.[1] This computer is the central processing unit of the Stretch system[2] contracted for development and delivery to the Atomic Energy Commission for their Scientific Laboratories in Los Alamos, New Mexico.

The purpose of this paper is to describe the major functions of the I-unit, give a general picture of the internal machine organization and the logical reasons behind it, and present several examples of how some of the performance goals were achieved.

A diagram of the Stretch system appears in Figure 1. It consists primarily of the central computer, 6 blocks of 2-usec memories (each containing 16, 384 words of 72 bits), a basic I/O Exchange unit with its associated I/O units and adapters, and a high-speed Exchange (disk synchronizer) unit with its disk unit and adapter. The system is expandable up to a maximum of 16 memory blocks, 32 basic I/O channels, and 32 disk units.

The computer is divided into five major elements: the memory bus control unit, the instruction unit, the lookahead unit, and the serial and parallel arithmetic units. These are shown in Figure 2.

One of the principal factors in achieving the high performance in the computer is the ability of these separate logical areas within the computer to operate independently and simultaneously. This means that while one of the arithmetic units is busy executing an instruction, the lookahead unit can be "stacking" up the following instructions with their operands, and the I unit can be fetching and indexing more instructions preparatory to loading them into lookahead. In many cases, the I unit can actually be executing an instruction wholly within the I unit, simultaneously with the execution of a preceding instruction in an arithmetic unit. In effect, the computer is a form of "pipe-line" which once filled is capable of a very high output rate. Basically, the main job of the I unit is to keep this "pipeline" filled by maintaining the instruction fetching and preparation rate compatible with the operating rates of lookahead and the arithmetic units.

A complete description of the specific functional responsibilities of the I unit will be given later. However, there were a number of more general requirements which significantly affected the entire design of the unit. The most important of these were:

1. It had to handle a large diversified instruction set in a wide variety of word formats.[3] (see Appendix A).
2. It had to prepare half-word instructions, full-word instructions, and full-word instructions across memory word boundaries.
3. To achieve the desired performance, it had to process several instructions simultaneously.
4. It had to be completely interruptable and recoverable on every instruction.[4]
5. It had to test for many exception conditions, set corresponding indicators, and be capable of suppressing or terminating the associated instruction if necessary.
6. It had to differentiate between three types of memory -- external memory (EM), index storage (XS), or internal register (IR) -- on all fetches and stores (see Appendix B).
7. It had to update the time clocks every millisecond.
8. It had to be virtually instantaneously stoppable to provide meaningful automatic error scans.
9. It had to be constructed with standard circuits, panels, and frames.
10. It had to be a reliable and thoroughly checked unit.

As a result of these requirements, plus the many assigned program functions, the instruction unit was designed and built to occupy five full double-gate standard frames and parts of three others, filling 46 standard panels, using approximately 1275 standard-circuit double cards and 6385 standard-circuit single cards, and requiring about 53,680 transistors.

Functions

The instruction unit has as one of its two primary functions the fetching and preparation of every instruction executed by the computer. The fetching carries with it the responsibility for checking and possibly correcting the word after it is received from memory. Every word in external memory contains 64 data bits and 8 error correction code (ECC) bits. These ECC bits permit single error detection and correction plus double error detection. The preparation of each instruction involves the indexing[5] of the instruction (if required), the partial decoding to determine instruction class and unit destination within the computer for execution, plus the actual operand fetch and loading of the instruction into lookahead. It also requires various tests of each instruction for indicator setting and possible suppressing and/or interrupting. Some indicators require that the instruction not be executed (suppressed), while others permit full or partial execution with the option of causing an automatic interrupt at the completion of the instruction. The actual point of execution and interrupt test is not until some time after processing by the I unit. In order to know the memory location of every instruction as it is tested for an interrupt, the I unit loads the advanced instruction counter (IC) value into lookahead with each instruction. If an interrupt is detected later, the program can store the IC value of the instruction following the one being interrupted. This enables the interrupt sub-routine to know where to return to the main program after the interrupt.

Two types of indexing can be specified -- normal and progressive. The normal mode modifies the operand address by the value of the index word with the resultant effective address replacing the original operand address. In progressive indexing, the result of the algebraic addition of the operand address and index value replaces the index value, and the original index value replaces the operand address as the effective address. This mode can also specify a stepping of the count field and may or may not call for an automatic refill if the count goes to zero.

All instructions eventually are loaded into lookahead. There are four levels of storage in lookahead, and each contains an op-code field, an indicator field, an operand field, and an instruction counter field. Normally an instruction only requires one level of lookahead; however, some require more. An example is a variable field length instruction in which one level is used entirely for operation definition, such as variable field length, byte size, and offset. A second and possible third level is then required for the operands.

All instructions to be executed by the floating point (FP) unit, the variable field length (VFL) unit, and the Exchange units are loaded into lookahead to await operand return and subsequent execution. As the instruction is loaded, the operand is fetched to the lookahead (LA) operand field, any indicators set by the instruction so far are transferred to the LA indicator field, and the IC value for the next instruction is set into the IC field.

If an instruction is of the index arithmetic type which is executed within the I unit, lookahead is loaded at the completion of the execution. In this case, the lookahead operand field contains the old contents of the index word that was modified. The indicator and IC field are set normally. This provides a means of recovering the index words which are modified out of sequence should an interrupt occur on a previous instruction.

The second primary function of the I unit is the actual execution of a large number of instructions in the Stretch instruction set. Index arithmetic instructions form the largest class in this set and they include direct, immediate, and indirect forms of addressing. The direct address refers to a location in memory for the operand, whereas the immediate address is the actual operand. There are several special instructions which act upon the index registers. Load Value Effective is a load type of instruction using indirect addressing, where the contents of the location in memory specified by the address may refer to another memory location. Load Value with Sum provides a means of multiple indexing by utilizing a form of geometric addressing where each bit of the address refers to a separate index register. Rename provides a method of "naming" index registers by putting in one index register the location in memory from which the contents of another index register came and to which it can be returned.

The Branch instructions comprise another class which the I unit is responsible for executing. They include all unconditional and conditional branches. The conditions may depend upon: any of the indicators, any bit in memory, and any index count field. In addition, each instruction has a number of modifier bits which specify whether the branch should occur on a zero or one condition, whether the bit should be left alone, inverted, set to zero, or set to one, and in the case of index count branching whether the value field should be modified by 0, +1, +1/2, or -1. The index and indicator branch instructions are half-word in length and the branch on bit is a full word. The half-word instructions may have a half-word prefix specifying a store instruction counter operation if the branch is successful. This forms a full-word instruction. The store instruction counter half-word instruction never occurs alone.

If an indicator branch is conditional upon an index indicator, the I unit determines the status and completely executes the instruction. If the condition depends upon some other indicator, then the I unit completes the instruction, assuming the branch to be unsuccessful. It loads the necessary operation code, indicator location, and recovery information (branch address) into lookahead to cause a test of the indicator later, after all previous instructions in LA have been completely executed. If at this time the branch is found to be successful, then a branch recovery operation is initiated and lookahead returns the branch address and all old index words to the I unit. . This recovery is similar to an interrupt recovery.

Word transmission instructions form another class which the I unit must execute. These include two basic types. One is the Transmit, which transfers data from one location in memory to another. The other type is Swap, which causes an interchange of data between two locations in memory. These instructions also contain a modifier bit which specifies whether the word count is contained in the instruction (immediate) or in an index word (direct). Another modifier specifies whether the addresses are stepped forward or backward.

The remaining instructions executed by the I unit include a general Refill and a Refill on count zero which provides the ability of refilling any word in memory. Execute and Execute Indirect are two instructions which provide the ability to execute "subject" instructions at direct or indirect locations in memory. The Store Zero instruction forces zeros into any memory location. A complete list of I-unit instructions appears in Appendix C.

Another function directly involving the I unit is the automatic interrupt operation. At the completion of every instruction execution, a test must be made on the indicators to determine if an interrupt is required or not. If not, normal program operation is continued. If an interrupt is called for, recovery operations are initiated in lookahead and the I unit An interrupt is signaled whenever the interrupt system is enabled (by the program), and an indicator in the register and its corresponding bit in the mask register are one's. The I unit must determine the indicator causing the interrupt, reset the indicator, and locate the "free instruction " associated with that particular indicator. It then fetches the instruction, prepares it, and either executes it or loads it into lookahead. If it is not a successful branch instruction, the I unit returns to the original program and continues normal operation; hence the term "free instruction". However, if it is a successful branch, the I unit branches to the new program routine and proceeds normally until a new branch instruction returns it to the original program. The interrupt mechanism is automatically disabled during the fetching and execution of the "free instruction".

Time clock operation is another function assigned to the I unit. The computer contains two time clock values; an interval timer of 19 bits (8-1/2 minutes) and a real time clock of 36 bits (777 days). These two quantities are contained in one word located in the small, fast-access, index core storage. Approximately every millisecond (1024 cps), while the computer is under program control, the I unit must stop its normal operation, fetch this word, and step the two clock values through the index adder. The interval timer is stepped -1 and the real time clock is stepped +1. When the interval timer goes to zero, a corresponding indicator is set. The interval timer may be set by the program but the real time clock may not.

The I unit also has the responsibility for monitoring all memory addresses for out-of-bounds conditions. All external memory fetches for the computer are initiated by the I unit, and each address is compared against an upper and a lower boundary register. Depending upon the state of an outside-inside

control trigger, appropriate indicators are set according to the type of fetch (instruction or data). All stores are performed by the lookahead. However, before loading lookahead with a store operation, the I unit tests the store address and sets a store indicator in lookahead, if out of bounds. These indicators may cause suppression of the instruction and/or an interrupt and automatic branch to a corrective routine.

The last important function of the I unit is providing manual controls for direct intervention by the operator console and customer engineering maintenance console. Since the I unit has complete control over all operations to be performed by the computer, it was found to be the logical place for the majority of the manual controls. These controls provide the ability to:

1. Start or halt the machine.
2. Load new programs.
3. Display or store memory.
4. Single-step the program an operation or a cycle at a time.
5. Enter a particular instruction into the machine.
6. Put the machine in a repeat instruction mode.
7. Continuously test index storage.

Repeat instruction mode causes the machine to continually repeat the fetching, preparation, and execution of one particular instruction word.

In addition to these specific functions, several problems arose which required special handling. One of these was created by preaccessing instructions before completing the execution of previous instructions. In the case of store-to-memory types of instructions, it was possible that an instruction might be storing into the immediate program area and particularly into an instruction that had already been fetched by the I unit. To guard against this, a program store compare circuit was designed which compared all instruction fetch addresses against the store address register in lookahead. If the instruction fetch has not been made yet and it compares equal, the I unit waits until the store is complete before resuming. If the instruction fetch has already been made, an I-unit recovery must be made and the instructions refetched after the store is completed.

There are three different types of addressable memory in the system: internal transistor registers, index core storage registers, and the main external memories. This presents a problem in the I unit. When indexing instructions and fetching operands rapidly, the I unit loads lookahead and fetches an operand on the cycle following the index modification cycle. This leaves no time to decode which type of memory is involved and to select the correct control sequence. It is presumed that the address refers to external memory, and an external fetch is begun. The decoding is complete before the midpoint of the fetch cycle. If the presumption proves to be wrong, the actual fetch can be blocked and the correct fetch control can be selected for the next cycle. If the presumption is correct, the original fetch is completed and a decode cycle has been saved.

These are just a couple of examples of the many complexities which faced the design group in designing the I unit to meet all the functional, performance, and reliability requirements of Stretch and at the same time keep the cost to a minimum.

Data Path Organization

The first step in the design of the unit was to design a data path system, with a minimum of hardware, that accomplished all the functions assigned to it. The next step was to determine the amount of time-sharing that was possible, and the amount of concurrency necessary to achieve the high performance goals desired, still at minimum cost. Finally, after arriving at a satisfactory compromise of the first two points, the system was studied carefully and checking/correcting circuitry added to obtain the high degree of reliability desired.

The result was a machine organization as shown in Figures 3 and 4. Figure 3 is a diagram of the data paths for the principal part of the I unit, and Figure 4 is a diagram of the interrupt mechanism. The basic I-unit organization consists of six transistor registers, two adder units with checking, six data paths, two address busses, and one checker/corrector unit. In addition, there are indicator and tag storage positions, a boundary compare unit, a program store compare unit, lookahead load data transfer busses, and a leftmost one detector-encoder for the multiple indexing Load Value with Sum instruction. The interrupt mechanism consists of two transistor data registers, one unit address register, and a leftmost one detector-encoder. These data paths account for approximately 40 per cent of the hardware. The remaining 60

per cent is taken up by extensive control logic, with its associated timing, and decoder circuitry.

The six main registers in the I unit are the instruction counter register (ICR), the instruction-data word buffer registers (1Y and 2Y), the index register (XR), the preparation and execution register (ZR), and the multi-purpose working register (WR).

Instruction Counter (IC)

The IC system was designed to provide the actual memory address of the instruction currently being executed or prepared for lookahead in the Z register, and at the same time fetch succeeding instructions into the Y registers. Since a number (6) of instructions may be located in the Y and Z registers simultaneously, some means had to be developed to keep track of the instruction addresses as the instructions proceed through the I unit. It was found that we could eliminate the necessity of multiple IC registers by using the one ICR to keep track of the instruction being operated on in the ZR, and using outputs of the ICR or the IC adder to fetch following instructions.

The ICR contains 21 bit positions, two of which are parity bits. The two low-order bits are separated from the seventeen high-order bits and have their own individual advancing mechanism to provide the flexibility needed for advancing by half and full words. The high-order seventeen (0-16) bits feed a plus one, a parallel, carry propagate adder, which actually provides the (ICR quantity +2) address. By selecting combinations of ICR and adder outputs, we can obtain four full word addresses: n, n + 1, n + 2, and n + 3, where n is the ICR address. This provides all the lookahead addresses needed for pre-accessing instructions into the Y-register instruction buffers.

The ICR has a set of in-gates (21) from the lookahead IC buffer for use in recovery operations. It also has a set of in-gates from the index adder out bus for branch operations. The final set of in-gates is for setting the IC adder output into the register for a full advance of the IC system.

Both ICR and the IC adder output can be gated out to the memory address bus for instruction fetching. The ICR also can be gated to the index adder in bus A (ABA) for store instruction counter and branch relative operations. The ICR has a complete set of ungated output lines to the lookahead IC input gates for loading the associated IC address into lookahead along with the instructions, and for program store testing. It has a set of ungated output lines to the IC adder, and several special lines to the control area. Another full set of these lines go to the maintenance console for indicating the contents of the ICR and the IC adder output.

Instruction and Data Buffer Registers (1Y and 2Y)

To achieve high-speed instruction preparation, particularly floating point instructions, it was necessary to provide two instruction buffer registers. These two registers are identical and are used alternately for receiving instruction words fetched from memory by the IC. They are also used for data operands required for the execution of instructions performed in the I unit. They each contain 73 positions; 64 data bits, 8 check bits, and 1 memory check bit. The 8 check bit positions may contain error correcting code (ECC) bits or parity bits. All words in memory contain error correcting code bits, but during the checking/correcting operation in the I unit, the ECC bits in the Y register are replaced with parity bits for checking internal operations. The memory check bit indicates whether or not an error occurred during the memory access, and, therefore, indicates whether the word received is valid or not.

The Y registers have two complete sets of input gates; one from the memory out bus (IMOB) for memory fetch returns, and one from the checker out bus (ICOB) for check/correct operations and internal word transfers.

Each Y register has a full set of out-gates to the checker in bus (ICIB), again for checking/correcting or internal word transferring. Each also has two sets of half-word (36 bit) gates to the adder in bus B (ABB) for direct transfer or arithmetic operation through the index adder. Four sets of out-gates from the two I fields of each register permit addressing index storage via the index address bus (XAB).

Both registers have 73 lines to the maintenance console for indication, plus a number of ungated lines to the control areas for instruction predecoding and special memory addresses.

The parity fields are split up across the Y registers in the following manner:

P_0 (0 - 17)	P_4 (32 - 49)
P_1 (18 - 23)	P_5 (50 - 55)
P_2 (24 - 27)	P_6 (56 - 59)
P_3 (28 - 31)	P_7 (60 - 63)

These fields were found to be the best combination for effectively handling all the different word formats encountered in the system.

Index Storage (XS) and Index Register (XR)

The specifications for the Stretch Computer called for 16 index words located in high-speed storage. The first approach was to use 16 transistor registers, but this was soon found to be undesirable for several reasons. One was that it was extremely expensive, particularly if each register was to be capable of directly operating with the index adder to perform all instruction indexing functions and the execution of all index arithmetic operations. Another reason was that the large amount of hardware involved presented a packaging problem and detracted from the anticipated high performance. It was apparent that a buffer register would be required, which alone would have all the required logical capabilities and would allow each of the 16 index registers to be transferred into it prior to execution. This still presented a large and expensive piece of hardware. The ideal solution was found by providing a compact, high-speed, 16-word, non-destructive-read, core memory for the index words, with one data register (XR) to read into, store out of, and perform all the logical operations required.

The index storage (XS) was actually designed with 17 words of 73 bits each. The seventeenth word contains the interval timer and the elapsed time clock for rapid access and advancing of these values. It is a two-dimensional array (17 x 73) and has a read-out time of approximately 200 nanoseconds. Total access time including address gating, transmission, and decoding requires one machine cycle. To store the contents of the XR into XS requires two cycles. The first one destructively reads the selected word, thereby resetting it to zeros, and the second cycle writes the XR contents into the selected row of cores. The index address is checked during the decoding. The XR is checked during the following logical operation for which the index word was fetched.

The index register was designed with two principal objectives in mind. One was to provide the function of a data register for fetching and storing to/from index storage. The other was to provide the ability of executing all the full word, half word, field transfer, and logical operations required to execute all the I-unit instructions. The result is that the XR has five sets of in-gates and out-gates with many separately controllable fields.

This register contains 73 bit positions including 64 data and 9 parity check bits. Eight of the check bits correspond to the eight in the Y registers, and the ninth is the parity on bits 46 - 49 to provide means of obtaining a parity check on the index count and refill fields.

The primary input to the XR is directly from the sense amplifiers of the index storage during an index fetch operation. A full set of gates (73) allow gating from the I-checker out bus straight into XR for checking and full word transfer operations. Four sets of gates allow gating from the adder out bus into four different fields of the XR. These fields are: the 25 bits of XR beginning at position 0, the 25 bits of XR beginning at position 32, the 18-bit count field beginning at position 28, and the 18-bit refill field beginning at position 46. These gates are all used in the execution of various types of index arithmetic instructions. In addition to these inputs there is a direct reset of all 73 positions for setting the XR to zero prior to a read-out of index storage.

The XR has a full set of out-gates (73) to the checker in bus for checking and full word transfer operations. It has three sets of partial gates to the adder in bus "A". These provide the ability of gating the value field, the count field, and the refill field to the adder for arithmetic operation or transfers through the adder. One set of gates (24-27) to the adder in bus B provides the ability to check the sign of the value field during value field operations in the adder.

There are three detector circuits connected directly to the XR. These circuits provide the following indications:

X value less than 0	X count equal to 1
X value equal to 0	X count equal to 0
X value greater than 0	X refill equal to all 1's

These are used to set indicators or modify execution controls for various operations. In addition, there are a number of special ungated outputs of the XR which feed adder true/complement controls, execution controls, and parity adjust logic in the parity checker/generators. A full set of ungated outputs go to the maintenance console for indicator purposes.

Preparation - Execution Register (ZR)

This register is the basic operating register of the I unit. Every instruction is placed in

this register from the Y registers for indexing, decoding, execution, and lookahead loading. It is full word in width to accommodate full word instructions and to speed up floating point half-word instructions. All full-word instructions appear straight, left to right in the register, regardless of how they were received from memory and placed in the Y registers. There are a large number of output gates on the register because of the many special functions performed in the register in indexing (normal and progressive), executing I-unit instructions, fetching operands, and loading the instructions into lookahead.

The ZR contains 74 bit positions, including 64 data and ten parity bits. Eight of the ten parity bits correspond to the standard eight in the Y and X registers. The other two are for parity on the channel address field (12 - 18) for I/O instructions, and the length field (35 - 40) of VFL instructions. Associated with the ZR is a small three-bit P register which is used solely for retaining the progressive indexing code (bits 32 - 34) during any index modification of the right half of a VFL instruction.

The only in-gates provided on the ZR are for gating the adder out bus into various positions of the ZR. There are two sets of these gates, one for the left half of Z and the other for the right half. These gates have split control to provide for partial gating. This takes care of all the in-gating required by instruction transfers from Y to Z, plus all arithmetic result gating into Z.

The out-gating of the ZR is more extensive and complicated. There are two sets of out-gates for gating the left half or the right half of Z to the adder in bus B for arithmetic operations on the left or right operand addresses. There is a separate gate for gating the length field (35 - 40) to the adder in bus A for word boundary cross-over test, and a separate gate for sending the immediate count (50 - 55) in transmit instructions to the W register for counting purposes. There are two sets of out-gates provided for gating the left (0-17) or right (32 -49) operand address to the memory address bus for operand fetches. There are five sets of gates for gating the left or right operand address, the left or right J field (index operand in index instructions), and the left I field (index address for index modification) to the index address bus. These permit fetching of index word operands and the index fetch for the delayed modification of the left half of Z.

There also is a set of ten out-gates to the checker in bus for rearrangement of fields for loading instructions into lookahead. These also have split control for selecting the width of the fields.

In addition to the in and out-gates, the ZR has 41 positions line driven to the control area for operation and memory area decoding. As in the case of all the registers, all positions of the ZR have an ungated line to the maintenance console for indicator purposes.

Working Register (WR)

This register is only 19 positions long (including one parity). It is used primarily for operand address storage, and secondarily as the counting register for transmission type instructions which cross memory word boundaries, for multiple indexing address decoding, and for automatic refill and interrupt address operations. In transmit operations the direct or immediate count field is placed in the W register for counting purposes, while the from and to operand addresses remain in Z for the fetching, storing, and stepping operations.

The WR has three in-gates: one from the adder out bus for transfer and arithmetic operations, one from the maintenance console keys for manual insertion of an address, and one from the interrupt bit address encoder for automatic interrupt operations.

The WR has three out-gates: one to the adder in bus A for arithmetic operations such as counting, one to the memory address bus for word boundary crossover and refill fetches, and one to the index address bus for index fetches.

Ungated outputs of the WR feed a leftmost one detect logical unit for multiple indexing address decoding. The output of the detect circuit feeds an address encoder, the output of which is set into a five-bit register called the geometric load address register (GLAR). The detect and encoder logic is checked.

Ungated outputs of the WR also feed various decoder circuits for determining special address and contents equal to one conditions. The output of the leftmost one detector (LMOD) feeds the adder in bus B for resetting the current multiple indexing address bit in WR by a subtract operation through the index adder. The output of the GLAR feeds the index address bus for index fetching during the Load Value with Sum execution.

All 19 positions of the WR and five positions of the GLAR go to the maintenance console for indicator purposes.

Index Adder Unit (IAU)

The specifications for the I unit called for arithmetic operations on fields up to 24 positions wide. High performance required a parallel adder of advanced design with a minimum of logical levels. In order to guarantee complete reliability, it had to be thoroughly checked. Since the many operations of the I unit required all the registers to be capable of feeding the adder, it was found that the transfer and adder paths could be combined and time-shared to provide the most economical and yet completely checked system. This was accomplished by providing an eight-bit bypass path around the 24-bit adder to permit half-word (32-bit) transfers. A further study indicated that the best performance could be gained by providing a controlled complementer on one input, with automatic re-complementing ability on the output. The basic 24-bit parallel adder was broken up into six four-bit blocks with parallel carry lookahead and carry propagate detect logic for each. Carries from block to block and end-around carries are detected early and propagated through. The 24-bit add or subtract is accomplished in five logical levels. Input checking is accomplished by comparing input parities with the half sum parity, and the rest of the adder is checked by a carry prediction checking method.

Standard I-unit parity is generated on the output in parallel with special memory address decoding. The bypass eight positions are parity checked and passed through to the output bus. The output is completely latched at sample time to prevent race conditions through the ungated paths when gating the result back into one of the input registers.

The adder in bus A (ABA) has a complementer on the input and is accomplished in the same logical level that does the ORing. The inputs to ABA are the IC, WR, XR, and ZR. The ABB is not complemented and the inputs come from the YR's, the LMOD, and the ZR. In addition, there are numerous lines to control the complementing, re-complementing, and parity adjustments for various fields and operations. The adder out bus (IAOB) has 32 data bit positions plus 11 parity bit positions for selection, depending upon the fields involved. The IAOB feeds the left and right halves of the ZR, the left and right halves, count, and refill fields of the XR, the WR, and the ICR.

The I Checker

One of the earliest requirements of the Stretch system was for automatic error correction of memory words. The method adopted was that of using the Hamming[6] error-correcting code (ECC) which required eight ECC bits with a 64-bit data word. In the I unit it was necessary to be able to check and correct memory words (instruction and operands) and convert to the I-unit parity system. It was also necessary to provide a full word transfer bus for high-speed transfer operations in executing many of the I-unit instructions. It was found that these two operations could share equipment by combining the transfer bus with the full word ECC checking/correcting and parity checking/generating logic. It was also found that the ECC checking/correcting operation on instructions could be overlapped with the initial pre-decoding required on the new instructions when they are received in the Y registers.

Lookahead had a similar problem with ECC checking/correcting and parity checking/generating on operands fetched to lookahead by the I unit, and in storing result operands to memory. A study of the two units showed that one I checker unit could be time-shared between the I unit and lookahead, provided a fast priority system could be designed to guarantee little loss in performance. This was done and the result is a single I checker with separate I checker in busses from the I unit and lookahead OR'd at the input to the checker. The checker out bus (ICOB) is a 64-bit data bus plus 29 parity and ECC lines for selection by the controlling unit for the proper parity or ECC bits for the receiving register. This bus feeds the four lookahead levels first and then the XR and the 2Y registers. The output of the checker is completely latched at sample time to prevent race conditions through the ungated paths while gating the result into the receiving register.

Interrupt Mechanism

Figure 2 shows a block diagram of the interrupt mechanism. It consists of a 64-position indicator register (IR), a 28-position mask register, and a leftmost one detect and encoder circuit.

The indicator register has two inputs; one from the arithmetic checker out bus (ACOB), and the other, the individual turn-on line from each indicator's particular logical area. These logical areas include the I unit, the VFL and FP execution units, lookahead, exchange, and memory. There is no parity on the contents of this register because it is continually changing, due to the many asynchronous inputs. The only out-gate on the IR is to the arithmetic checker in bus (ACIB) and is used for transferring the contents of the register to another location.

Similarly, the input gate from ACOB is for bringing in a new word to the IR. Ungated outputs from a portion of the register feed the updated indicator register in the I unit for recovery purposes. A full set of ungated outputs feed the leftmost one detector and the maintenance console.

The mask register has an input only from the ACOB for bringing in a new mask word. It also can be gated to the ACIB for storing purposes, and has ungated outputs to the leftmost one detector and maintenance console. Logically, the first 20 positions (0 - 19) of the mask register are always one, and the last 16 positions (48 - 63) are always zero. They are fixed and are not programmable. There are four parity bits associated with the mask register positions 21 - 47, and they conform to the parity fields of the arithmetic bus and checker.

The leftmost one detect circuit has two functions; one to test rapidly for any match between an indicator position and its associated mask bit, the other to determine, in the case of multiple matches, which one has higher priority. Priority is established from left to right (0 - 63), and the match with highest priority blocks the remaining ones from being effective. The test for any match is done in only a few levels by ORing all the compare And circuits, and signalling an interrupt if the mechanism is enabled. The enabling/disabling is controlled by a single trigger which is set on/off by programming. Once an interrupt is signalled, the leftmost one detect logic is allowed time to establish priority, to encode the matching pair of indicator and mask bits into the register bit address of the particular indicator, and set it into the WR. The bit address is a six-bit address plus one parity bit.

Included in this logical area is a seven-bit channel address register which is used to hold the address of the I/O unit which sets I/O status bits into the indicator register. This register may be set by either the basic or high-speed exchanges and includes two parity bits. It can be gated to the ACIB (positions 12-18) for transfer purposes.

Also designed for this area but not packaged was the other CPU register. This register of 19 positions is used for systems involving more than one computer. It can be gated out to the ACIB and gated in from ICOB for transfer purposes. There are four parity bits associated with the 19-bit field to conform with the arithmetic bus and checker requirements.

Updated Index Indicator Register (UXIR)

There are eight triggers in the I unit which contain the status of the index register involved in the most recent index arithmetic instruction executed by the I unit. These indicators differ from the status of the corresponding main indicator register triggers by the effect of the index arithmetic instructions which have been executed by the I unit but whose result indicators are still in lookahead awaiting transfer to the indicator register in proper instruction sequence-hence, the term "updated indicators". This UXIR is used to test for conditional branches on these indicators.

A listing of the indicators contained in the updated indicator register is as follows:

1. Index low - XL
2. Index equal - XE
3. Index high - XH
4. Index count zero - XCZ
5. Index value less than zero - XVLZ
6. Index value zero - XVZ
7. Index value greater than zero - XVGZ
8. Index flag - XF

Indicator and Address Tag Triggers

Due to the multiplicity of instructions that are contained in and being operated on simultaneously in the I unit, it is necessary to tag (store with the particular instruction) each instruction with identifying information regarding certain conditions which may arise during its processing. Examples of these conditions are memory and ECC checks on new instructions, parity checks on instruction transfers from Y to Z, and type of memory address decoded during the transfer through the index adder. These triggers are located for the most part in the control area, so that they can immediately condition the control trigger outputs of the following cycle. There are some 26 triggers of this nature in the control area of the I unit. These include the following:

A. Y Register Tags

1.	1Y instruction fetch indicator	1Y1F
2.	2Y instruction fetch indicator	2Y1F
3.	1Y operand address invalid	1YAD
4.	2Y operand address invalid	2YAD
5.	1Y identifiable check	1Y1DC
6.	2Y identifiable check	2Y1DC

7.	1Y memory check	1YMC
8.	2Y memory check	2YMC

B. Z Register Tags

1.	Z left operand address-special (0-15)	ZLSA
2.	Z left operand address-index (16-31)	ZLXA
3.	Z left operand address-nonexistent	ZLNA
4.	Z right operand address-special (0-15)	ZRSA
5.	Z right operand address-index (16-31)	ZRXA
6.	Z right operand address-nonexistent	ZRNA
7.	Z left instruction fetch indicator	ZLIF
8.	Z right instruction fetch indicator	ZRIF
9.	Z left operand address invalid	ZLAD
10.	Z right operand address invalid	ZRAD
11.	Z left contains identifiable check	ZLIDC
12.	Z right contains identifiable check	ZRIDC
13.	Z contains data store condition	ZDS
14.	Z contains data fetch condition	ZDF

C. W Register Tags

1.	W operand address-special (0-15)	WSA
2.	W operand address-index (16-31)	WXA
3.	W operand address nonexistent	WNA

D. General

1.	I unit contains non-identifiable check	NIDC

Control Organization

The remaining hardware in the I unit is taken up by an extensive and complex system which controls the flow of instructions and operands in the data paths previously described, in many different combinations of simultaneous and asychronous operations. The system consists primarily of many control triggers, commonly referred to as control stages or sequencers, plus their input and output switching logic and ORing of control lines to the data paths. Also adding to the hardware is the extensive decoder logic required to determine which operation and variation is called for in instruction preparation and execution. Associated with the control is a considerable amount of powering circuitry for the distribution of the clock pulses in each control area. These clock pulses originate at the computer master clock and are distributed to various units using delay line techniques for skew minimization. The data paths and control were designed to operate with a 4 megacycle clock and at present are operating at a 3.3 megacycle rate. The I-unit controls operate at half the master clock frequency.

The controls are divided logically into the following categories:

1. Instruction counter
2. Instruction preparation
3. Lookahead loading
4. Instruction execution
5. Miscellaneous

IC Controls

The instruction counter controls consist of eight control stages for sequencing the operation of fetching instructions to the Y registers, checking/correcting them through the I checker, and advancing the IC register. There are thirteen supervisory control storage triggers to condition the control stages as to whether a fetch is in progress (outstanding), 1Y or 2Y is empty, ZL or ZR is empty, a branch to 1Y of 2Y is required, or a recovery is required. Six tag triggers indicate whether there are any checks, instruction fetch alarms, or invalid addresses associated with the instructions in the Y registers. There are eight block and suspend triggers for instaneously interrupting the normal IC operation to allow the I-unit instruction execution controls the use of the Y registers.

These controls essentially attempt to keep the Y registers filled with new and checked instructions. They signal the preparation controls whenever Y register data is ready for transfer to Z. In branch and recovery operations, the IC controls are designed for rapid resetting and restarting at the new address in order to minimize the time required to refill the Y registers.

These controls time share the I checker and the memory address bus with other controls. They operate simultaneously with the preparation and lookahead load controls as long as no interlock occurs to indicate that no YR is empty, an I unit instruction requires execution, or an interrupt is required.

Preparation Controls

The preparation controls consist of eight control sequencers which control the preparation of all instructions for the computer. This preparation involves the transfer of instructions from the YR's to the ZR, the index fetching and address modification if required, and the word boundary crossover test for VFL instructions. In addition to the sequencers, there are six supervisory type control triggers which condition the selection of the right or left half of the current YR and the corresponding half of the ZR. Instruction pre-decoding as to type of instruction and indexing requirements is stored in eleven supervisory control triggers. There are eight additional tag triggers associated with the

two halves of the ZR to indicate the class of floating point instruction in Z, so that the lookahead load controls can rapidly take over and load lookahead without a delay for decoding purposes.

These controls signal the IC controls when a YR is empty, and signal the lookahead load controls or I-unit execution controls when the ZR has a prepared instruction. These controls are a completely independent set of hardware and operate simultaneously with the IC and lookahead load controls. Their function is basically to empty the YR's and prepare and fill the ZR as rapidly as possible, to insure a high rate of instruction flow through the computer.

Lookahead Load Controls

The lookahead load controls consist of fourteen sequencers which control the loading into lookahead of all I/O, VFL, and FP instructions.

Five of these deal solely with the loading of VFL instructions and are in the form of a five-stage execution timer. Every VFL load begins with the first stage and may then step to any one of the remaining four, so that every VFL instruction requires anywhere from two to five steps. Each sequencer loads a different lookahead level so that a VFL instruction may occupy two to five levels. If an operand address refers to an index address, the basic sequence is broken out of in order to fetch the index word by means of a common index fetch sequencer, and then control is returned to the VFL load sequencers.

There are two complete sets of our FP load sequencers for each half of the ZR. This was necessary to guarantee the fast switching and loading from the two halves of Z required to achieve the high FP performance. Each set of four FP sequencers controls the loading of one-level or two-level FP instructions, depending upon whether one or two operands are involved, and it controls the actual fetching of the operands from external memory, index storage, or internal registers. The majority of FP instructions require only one control step in which the operand is fetched from main memory at the same time that the FP instruction is loaded into the single lookahead level.

While this loading is being executed, the IC and the preparation controls may be simultaneously operating to maintain the instruction rate. The lookahead load controls signal the preparation controls when an instruction is completely loaded and the instruction in the ZR can be replaced with a new one.

Instruction Execution Controls

To execute the large number of I-unit instructions in all their variations required the design of a large control system. The instructions were categorized by type and then divided into logical operations and analyzed for maximum sharing of common controls. Circuit limitations and packaging rules often prevented sharing as much as was desired. As the control logic grew, it became necessary to package it in two frames instead of one, which, in turn, created a communication problem in critically timed areas. In order to keep the number of logical levels to a minimum, it was necessary to design a large amount of parallel logic into and out of each control stage.

The instruction set was divided into four categories: index arithmetic, branch, transmit, and miscellaneous operations. Each group was worked on separately, and the control sequences and logic required were designed. These controls were then compared for similarities, and wherever possible, common control logic was combined. The result was that sixty-one control sequencers and nine supervisory triggers, plus an operation decoder, were required to execute the instructions in satisfactory performance times. These controls provide for fetching and checking operands from memory (main memory, index storage, or internal registers), index adder logical operations, partial and full word transfer, checking operations, and the loading of lookahead with a large variety of operation codes and indicators. Most of these operations have a large number of input and output conditions whose combinations can cause each operation to be performed in a wide variety of ways. Other requirements of these controls were that they be able to stop immediately on errors, be recoverable in case of instruction suppressing and interrupt conditions, and be able to manually step a cycle at a time. Every control stage has a line to the maintenance console for indication.

Miscellaneous Controls

These controls are used primarily in the execution of the manual operations and the special recovery routines. They consist mostly of supervisory control triggers which initiate, condition, and terminate control sequences which perform the desired functions. The actual operations within the sequences are controlled by sequencers in the instruction execution area, and are shared for this purpose. For this reason the execution operation decoder must be blocked so as not to affect the stepping of the sequences for these special operations. Some special con-

trol functions also taken care of in this area are: a store wait control when lookahead contains a store to XS, or the indicator or mask registers; time clock operation triggers; and I unit recovery because of program store test. Some of these special functions are very critically timed, since they must block normal operation immediately, or it will be too late and unrecoverable damage may be done. These controls are packaged in the IC control area so as to minimize the communication delay of the interlocks.

Performance Characteristics

Since the Stretch computer was originally contracted for by the Los Alamos Atomic Energy Commission, certain of its performance goals were particularly important to the customer. Of primary importance was the complete floating point operation, including the instruction fetching, preparation, operand fetching, and actual execution. In the branch instruction, the Count and Branch instruction was to be used extensively for controlling the many iterative program loops characteristic of scientific computing. The variable field length operation was desirable and attractive but its performance need not be, nor could it be economically, as high as FP. The index arithmetic instructions were all very important.

The manner in which the Instruction unit achieves the desired performance goals in these areas will follow. These examples are chosen because of their special importance in scientific computing, and one should not infer that all the remaining operations were not important or not at a comparable performance level. The high overall performance goals of Stretch required that all the operations be executed in a much more powerful manner than any previous machine.

Floating Point Instruction Preparation

A timing diagram showing the preparation of continuous FP instructions is shown in Figure 5. The preparation includes the fetching of instruction words from 2 usec memory, the ECC checking of the instruction, the Y to Z transfer and index fetch, the address modification, and finally the operand fetch and lookahead load. It can be seen how successive instruction fetches, preparations, and lookahead loads are overlapped to achieve a performance goal of one FP instruction every two cycles. Each instruction is assumed to require indexing. If this were not the case, then instantaneous rates would reach one FP instruction every cycle, but the average rate would still remain at one per two cycles. This is because the maximum rate of instruction fetches is balanced with the maximum indexing rate and lookahead rates.

The diagram starts out by assuming a start-up operation (either program start or recovery operation) where two rapid fetches are made by the IC to the Y registers. When the words are received from memory, they are immediately checked; if there are no errors, an extra correct cycle is not required. During the check cycle, the ECC is converted to parity and the preparation controls pre-decode the type of instruction. In this case each word contains two FP instructions. The next cycle is the transfer from the left half of 1Y to the right half of Z. The criss-crossing of half-word locations was adopted as the simplest way of handling the different combinations of full-word and half-word instructions that can occur. During this transfer, any addressed index word is fetched into the XR. Assuming each instruction is to be indexed, the next cycle is the actual address modification, where the index value in X is added to the operand address in ZR through the index adder and the result replaces the original operand address. These last two cycles tied up the IAU, so the next instruction in 1 YR waited. The first instruction is now ready for loading into lookahead and fetching of the operand. With the IAU not busy, we can also transfer the next instruction to ZL and fetch its index word. The operand fetch would be initiated were it not for the 1 YR-ZL transfer emptying the 1Y register. The IC controls anticipate this and try to fetch the next instruction word (IC + 3) into it. This conflicts with the operand fetch of the lookahead load cycle, but since the IC controls have priority, the IC fetch is made simultaneously with the 1 YR transfer to Z and the index fetch. The lookahead load cycle is blocked during this cycle and completes the operand fetch and the load operation on the following cycle, overlapped with the modification of the second instruction. This early instruction fetch guarantees that the 1Y register will be filled and checked by the time the 2Y register is emptied. In this manner the flow of instructions can be maintained. The degree of simultaneity in the I unit is best illustrated by the cycle with the single asterisk (*) which shows an instruction fetch to 2Y, a check cycle on 1Y, a 2YR transfer to Z, an index fetch, and an attempted lookahead load, all occurring at the same time.

The high degree of overlapped operation in the computer is best shown by the last cycle (**). In this cycle the following operations are

occurring simultaneously:

1. Instruction 1 is being executed and checked.
2. Instruction 2 is being transferred from LA to the PAU.
3. Three of the 4 LA levels are loaded with instructions 2, 3, and 4.
4. Instruction 5 is ready to be loaded and its operand fetched.
5. Instruction 6 is being transferred from 1YR to ZL.
6. The index word required by instruction 6 is being fetched.
7. Instructions 7 and 8 are being ECC checked in the I checker and being pre-decoded.
8. The IC is fetching the 5th instruction full word containing instructions 9 and 10.

This is but one variation of FP preparation. Other variations develop when the operand address refers to index storage or an internal register, or where a second operand is implied, as in Multiply Cumulative and Load Cumulative Multiplicand, or when different types of instructions are intermingled with the FP instructions.

The diagram not only indicates how one performance goal is achieved, but also indicates the large amount of control complexity required to efficiently interlock and execute this I-unit instruction preparation function. It further illustrates how complete overlap of instruction fetching, indexing, and lookahead loading is achieved.

VFL Instruction Preparation

The preparation of a variable field length instruction is shown in the timing diagram of Figure 6. Starting at the same point as Figure 5, it shows the sequence when the second instruction is a full word VFL instruction located across memory word boundaries. In this case the left half of the first instruction word fetched is an FP instruction, and the right half is the left half of the VFL instruction. The left half of the second instruction word fetched contains the right half of the VFL instruction, while the right half of this word contains another FP instruction.

The first instruction is processed exactly the same as in the previous example. Notice that this time when the right half of 1Y is transferred to Z left, the eventual correct alignment of the full-word instruction in Z is provided for. Again, any index word addressed by the half word in 1YR is fetched into the XR. Before proceeding into the modification cycle, however, the type of indexing called for, normal or progressive, must be determined. This information is contained in the right half of the instruction, and is not available until the 2Y register has been filled, checked, and pre-decoded. In the example, this is completed by the end of the 1YR to ZL transfer, so no wait is required. Assuming normal indexing, the next cycle is the modification cycle for the left half. If progressive indexing (PX) had been specified, a much different sequence would have been required, and the operation would be done in two steps. The first one would replace the VFL operand address with the value field of the index word, and the instruction preparation would continue on and be loaded into lookahead. Following the lookahead loading, a control sequence would be entered where the increment, count, and refill operations on the index word, if called for, would be executed. The results of this operation would then be loaded into lookahead as the second of PX operations.

After normal modification of the left of the VFL instruction, the right half is transferred from 2YL to ZR and its index fetch executed. The next cycle then modifies the right half of the instruction. The indexing and instruction operation codes are preserved during the modification. The next step is to determine whether one or two memory words are required to obtain the operand field. The operand field can start at any bit position in a memory word and extend into the next memory word as long as the total field length is 64 bits or less. The word boundary crossover test (WBC) is done by adding, in the index adder, the length field to the full 24-bit operand address field. If a carry into position 17 of the operand address field occurs, then the instruction operand does, in fact, cross memory word boundaries. The high-order 18 bits of the result are gated into the WR, and this is actually the operand address for the second memory word.

This completes the preparation, and all that is left is to load the instruction into lookahead and fetch the operands. In this case three cycles are required. The first loads the instruction operation information into one lookahead level. Since the data field is used for part of the instruction information, no fetched operand can accompany this level. The next two cycles are for fetching the two operands of the instruction into two more levels of lookahead. The number of cycles required to load VFL type instructions varies from two to

five, depending on whether the instruction requires one or two operands, whether it is a fetch or a store to memory type of instruction, and whether any special registers are implied in addition to the operand address.

Having successfully loaded the instruction, the next half word FP instruction can be transferred from 2YR to ZL and the normal FP preparation continued.

The diagram shows that the VFL instruction in this case took eight cycles as compared to two cycles for a normal FP. If the instruction had no indexing specified and only one operand memory word was required, the operation would have only taken five cycles. Again, there are many variations of these instructions, depending on the type of indexing required, whether the instruction arrives straight or across memory words, whether the operands are in XS, EM, or IR, and whether it is a store to memory operation or not.

Count and Branch Execution

The execution sequence for a Count and Branch (CB) instruction is shown in Figure 7. Assuming a continuation of the FP preparation of Figure 6, the fifth instruction is defined as a CB instruction. In the example, it is assumed that the instruction requires indexing, and therefore, it is possible to overlap the operation decoding with the modification cycle. Otherwise, it would have been necessary to take a separate decode cycle. Once decoded, the operation enters an execution sequence controlled by the decoder outputs and conditions arising out of the individual operations. The first cycle is a fetch of the index word, whose count field determines whether the branch is successful or not. The next cycle, the fetch of the branch address instruction, is initiated, and at the same time the count field of the index word in the XR is decoded for a XC = 1 condition. If the condition for branching exists, the fetch is completed; if the condition does not exist, the fetch is blocked. This permits as rapid a fetch of the next instruction as possible, in order to begin filling up the Y registers with the new sequence of instructions. Overlapped with the instruction fetch is the loading into a lookahead level of the index word before modification. This is referred to as a psuedo-store level, and is only used in recovery operations where the index registers have to restore to some previous level as a result of an interrupt or no-op condition. The next cycle is the actual counting down through the index adder of the count field of the index word in the XR. After this is done, the value field can then be advanced or diminished by one or one-half, if called for by the instruction. This is also done through the index adder. At the same time, the advanced IC value (address of the instructions following the CB) is loaded into the same lookahead level as the psuedo-store. This is done because the next cycle destroys the IC contents by transferring into it the branch address from Z. If a no-op of this instruction is required, the I unit can continue straight on in the program. After the advance/diminish cycle, the updated index word is returned to XS. This is done by a clear cycle for resetting the word during the same cycle that transfers the branch address into the IC, and then following with a store cycle which sets the contents of the XR into the previously reset word in the array. To complete the operation, all indicators associated with this instruction and the new IC value are loaded into lookahead to be tested later in proper instruction sequence for an interrupt. If a condition occurs which will no-op the instruction, then the new IC field is not loaded, only the indicators and a no-op tag. This will cause any branching already done to be cancelled by a recovery operation later.

Since the instruction branched to was fetched early in the operation, the new instruction word has arrived and has been checked by the time the CB instruction is completely executed. Immediately the normal instruction preparation is restarted, and in this case, the branched to FP instruction is loaded into lookahead four cycles later.

The entire execution of the instruction took six cycles and was overlapped with the operand fetch, ECC check, and execution of instruction 1. In other words, it took ten cycles from the completion of one instruction in the program to branch to another program location and prepare and load the first new FP instruction into lookahead.

Variations of this instruction execution sequence occur with respect to the branch condition being 0 or not 0, the advance/diminish modifiers, and the setting of indicators that may cause a no-op or interrupt. In addition, there are other instructions in the same category which are more complicated and time-consuming. These include the Store Instruction Counter if Count and Branch (SIC - CB), the Count, Branch, and Refill (CBR), and the SIC-CBR variations.

Summary

The instruction unit is a large complex, high-speed computer unit designed and built to

provide the major functional ability and control for the Stretch Computer. A complete description of the major functional requirements is given along with some examples of the difficulties encountered. The machine organization, including data paths and controls, is described, and many of the primary design considerations are discussed. The complexity and size of the unit are largely determined by the instruction buffering, fast access index registers, the extensive overlapping and simultaneity of operations, and the innumerable combinations and variations of instruction sequences that have to be controlled. The control is achieved by a synchronous clock controlled network of variable sequence execution control stages. The performance is shown for a few typical and particularly important operations.

Acknowledgments

To give individual credit to the many people who have contributed to the design of the instruction unit would be impossible. However, major design contributions were made by the following individuals: Mr. S. F. Anderson for the index adder and branch instruction execution controls, Mr. L. L. Headrick for the interrupt mechanism, index arithmetic, and transmit instruction execution controls; Mr. C. R. Holleran for the IC system and lookahead load controls; Mr. S. L. Lindauer for a portion of the data paths and the instruction preparation controls; and Mr. L. F. Winter for his assistance in the design of the data paths.

The overall engineering effort was under the supervision of Messrs. E. Bloch and R. E. Merwin.

References

1. Erich Bloch, "The Engineering Design of the Stretch Computer," Eastern Joint Computer Conference Proceedings, December, 1959.

2. S. W. Dunwell, "Design Objectives for the IBM Stretch Computer," Eastern Joint Computer Conference Proceedings, December, 1956, p. 20.

3. W. Buchholz, "Selection of an Instruction Language, "Western Joint Computer Conference Proceedings, May, 1958, p. 128.

4. F. P. Brooks, Jr., "A Program-Controlled Program Interruption System, "Eastern Joint Computer Conference Proceedings, December, 1957, p. 128.

5. G. A. Blaauw, "Indexing and Control-Word Techniques," IBM Journal of Research and Development, July, 1959.

6. R. W. Hamming, "Error Correcting and Error Detecting Codes," Bell System Technical Journal, 1950, pp. 29, 147.

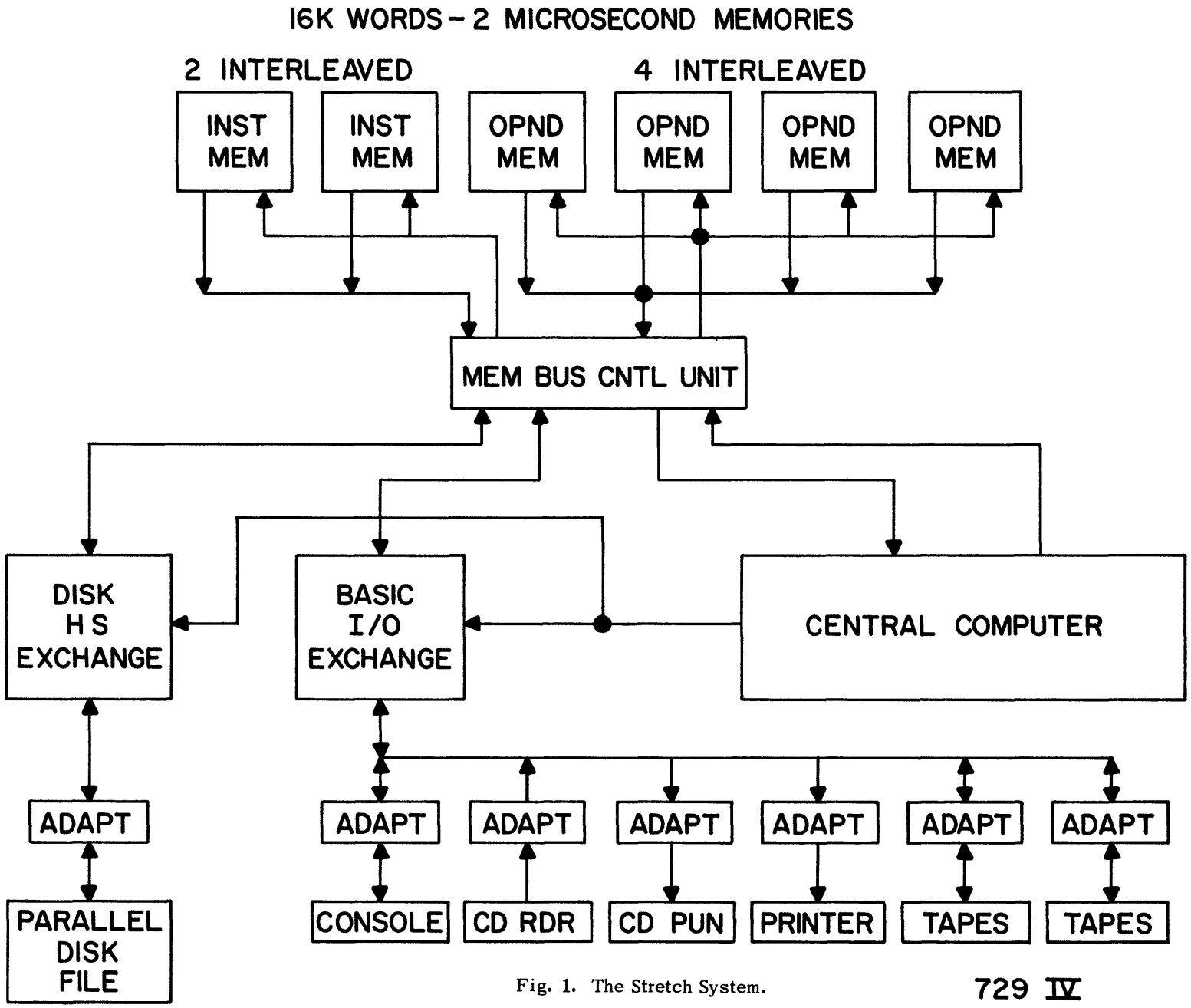


Fig. 1. The Stretch System.

729 IV

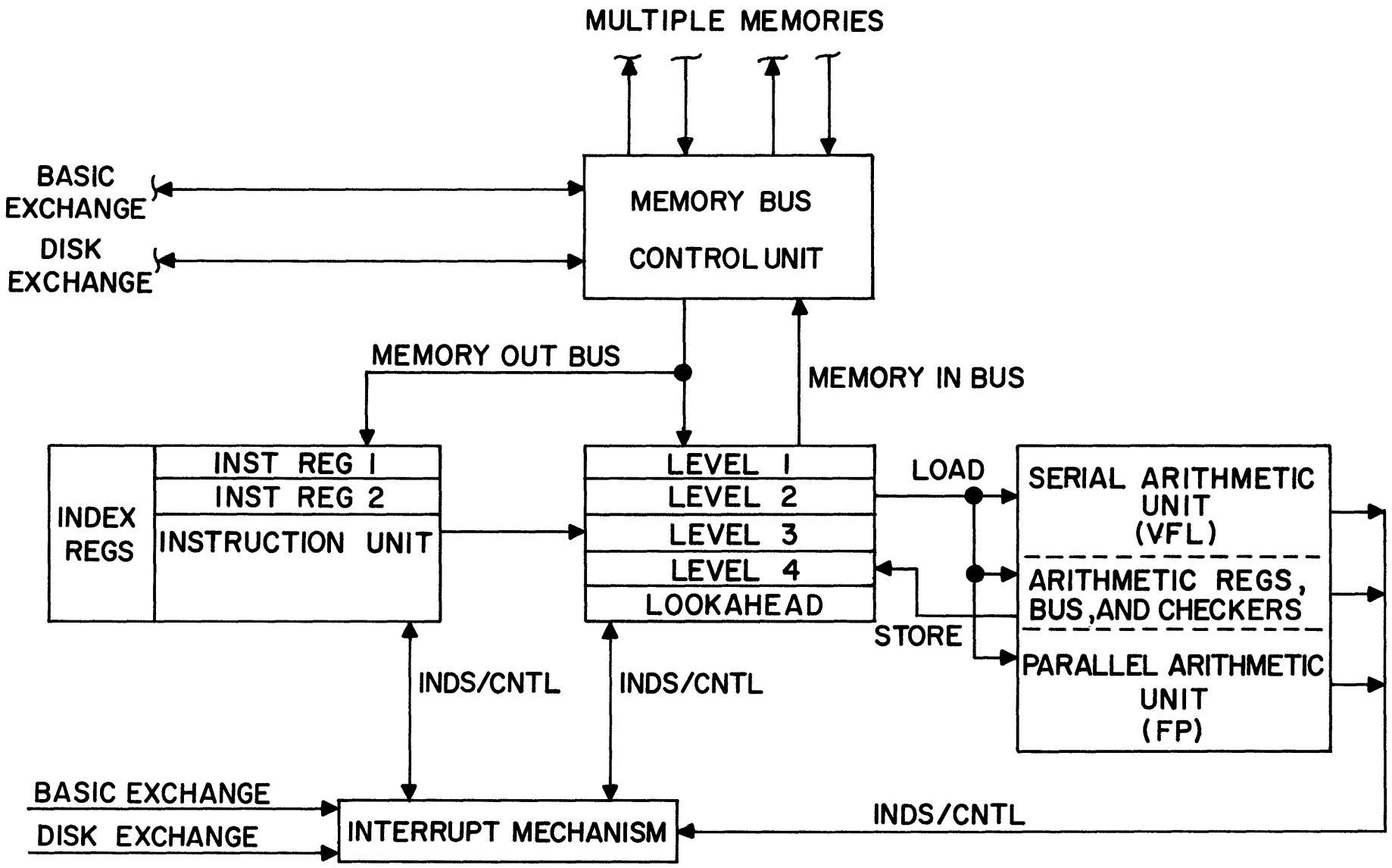


Fig. 2a. The Stretch Computer Central Processing Unit.

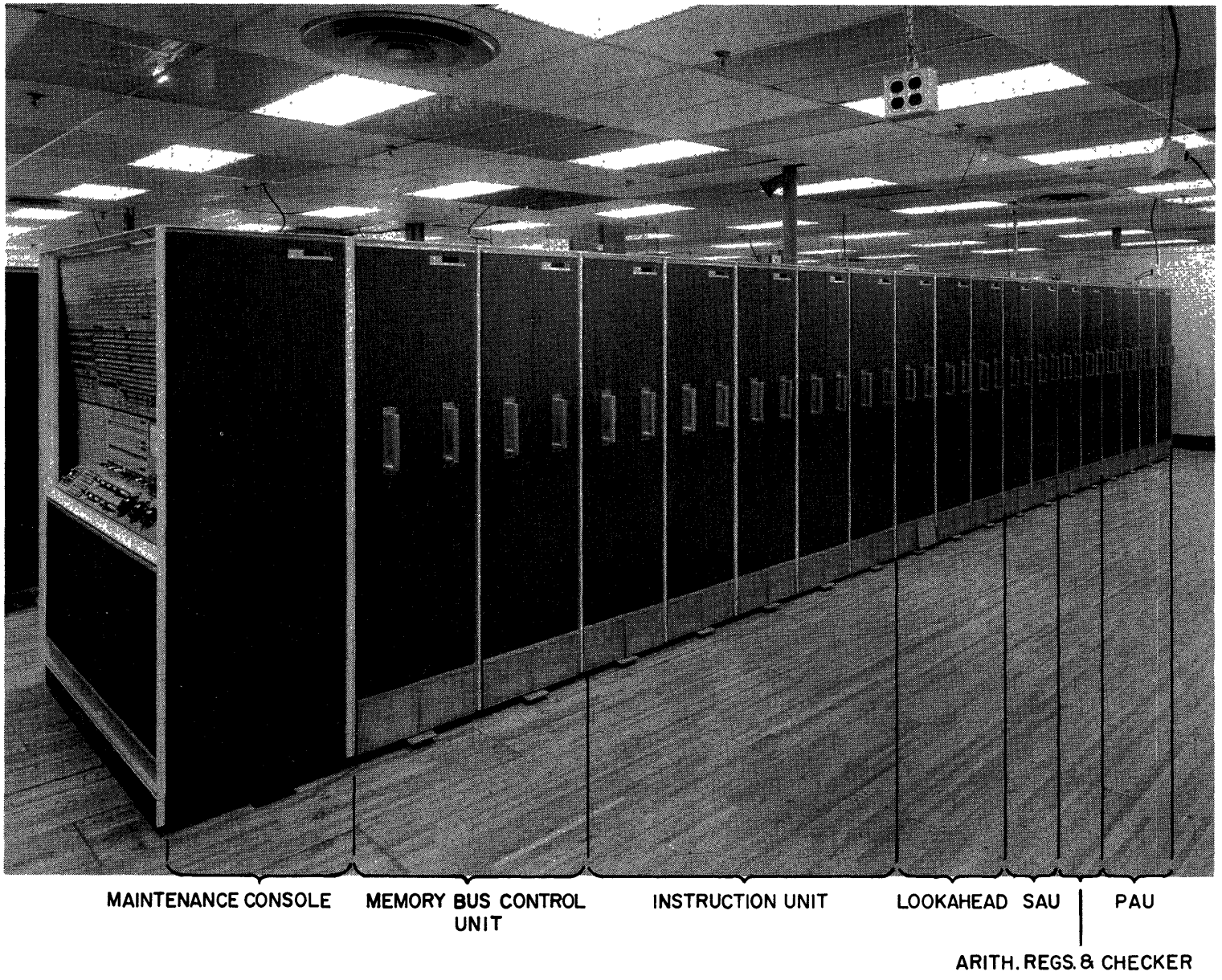


Fig. 2b. The Stretch Computer Central Processing Unit.

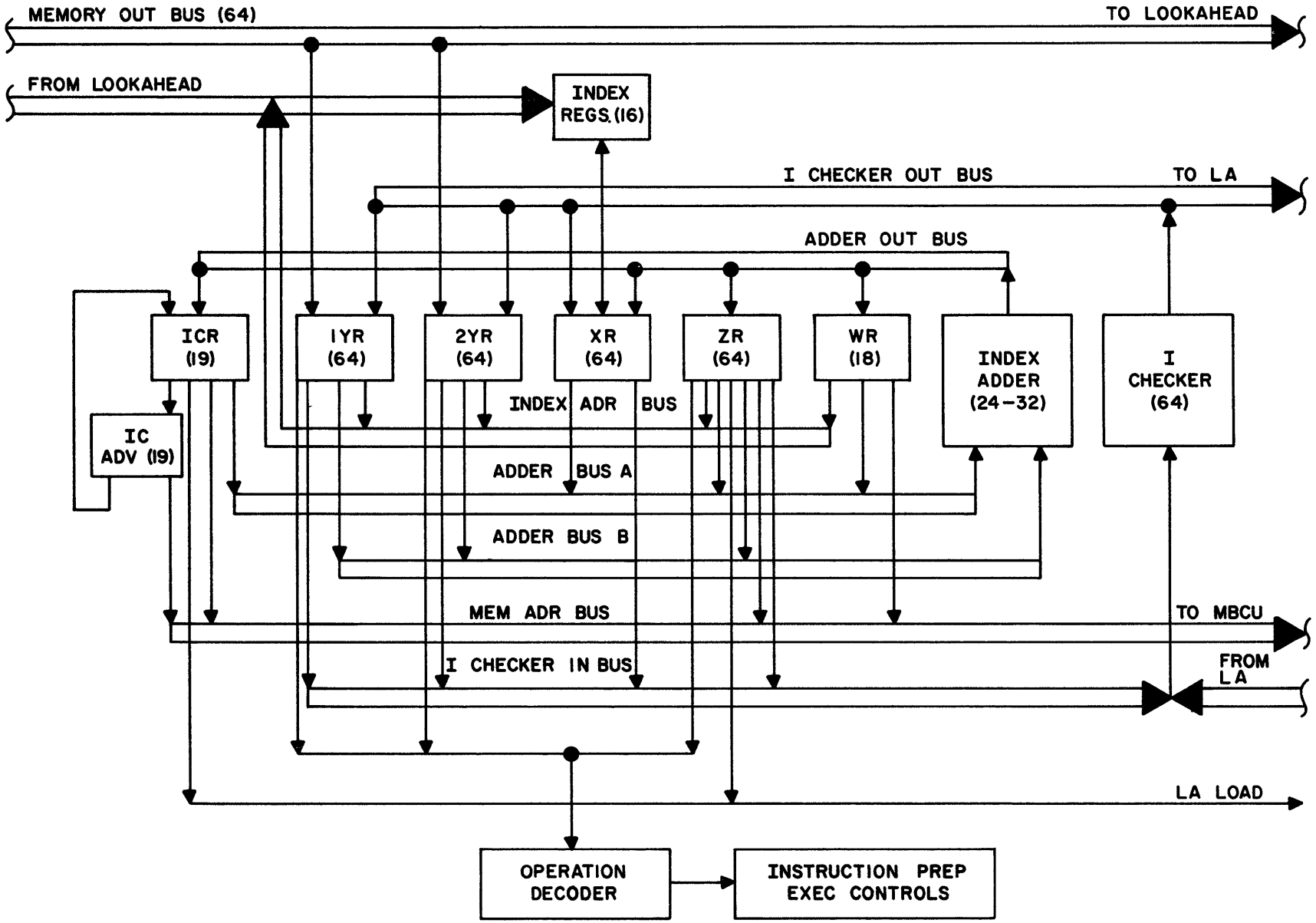


Fig. 3. Instruction Unit.

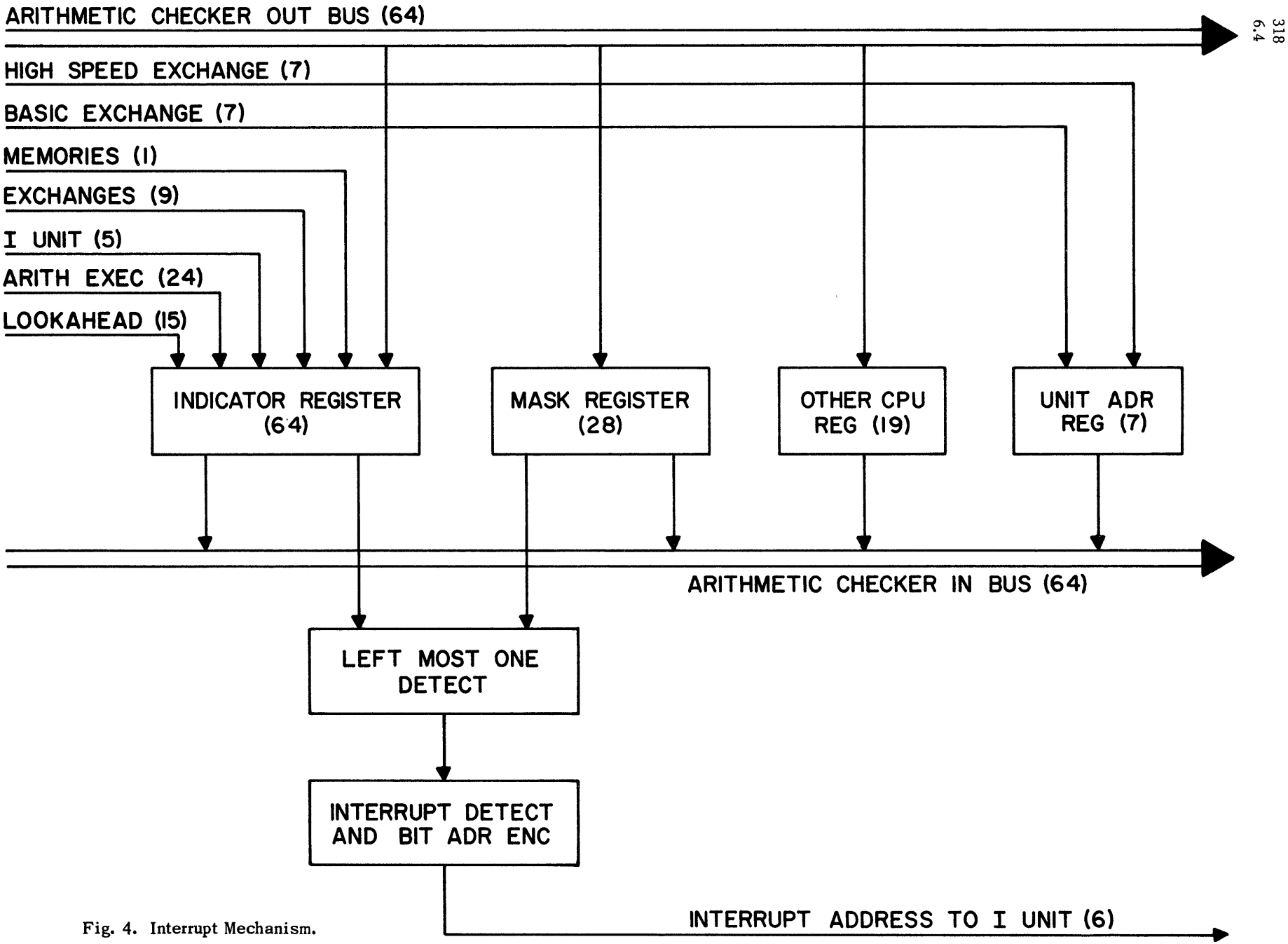


Fig. 4. Interrupt Mechanism.

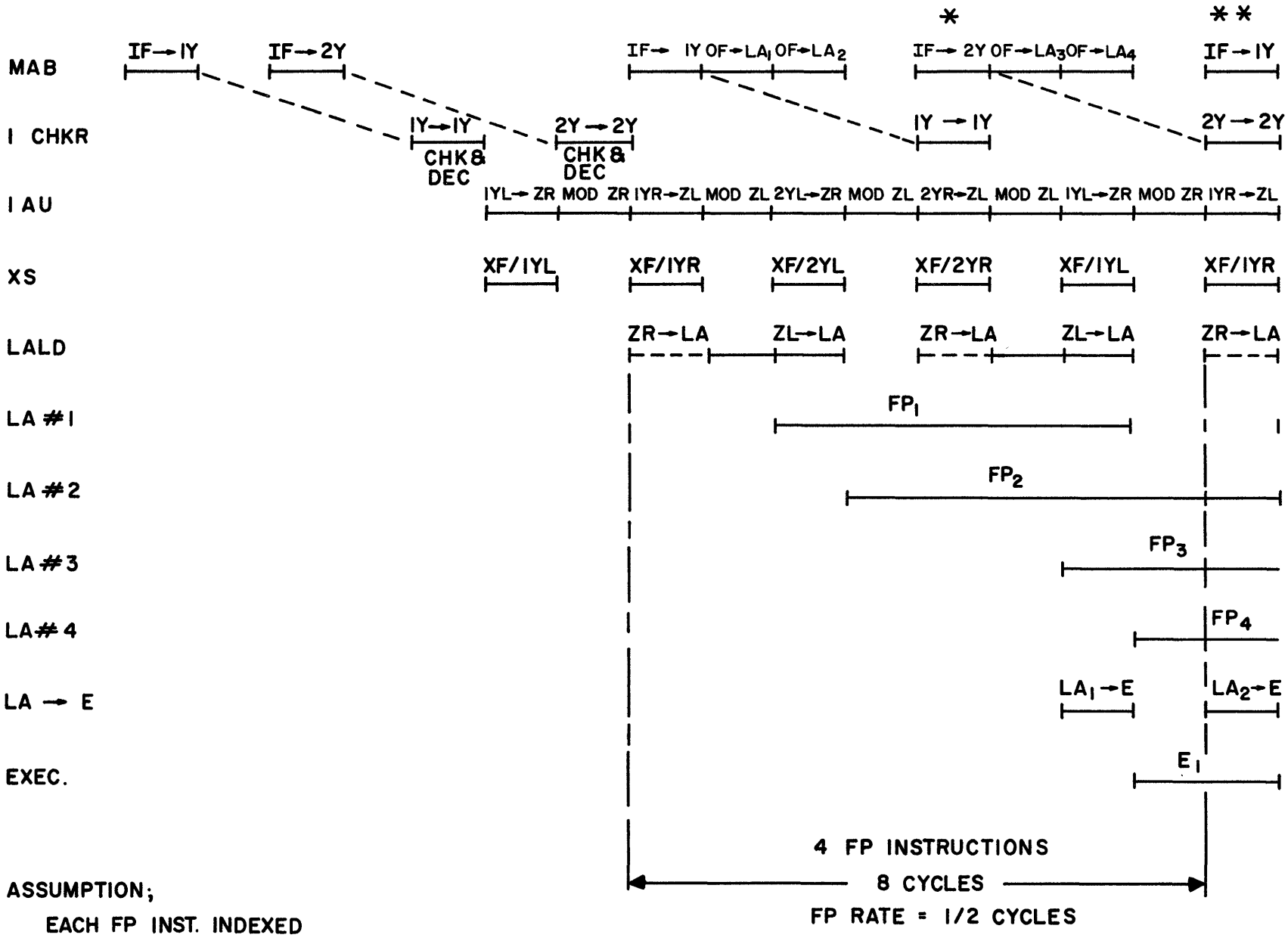
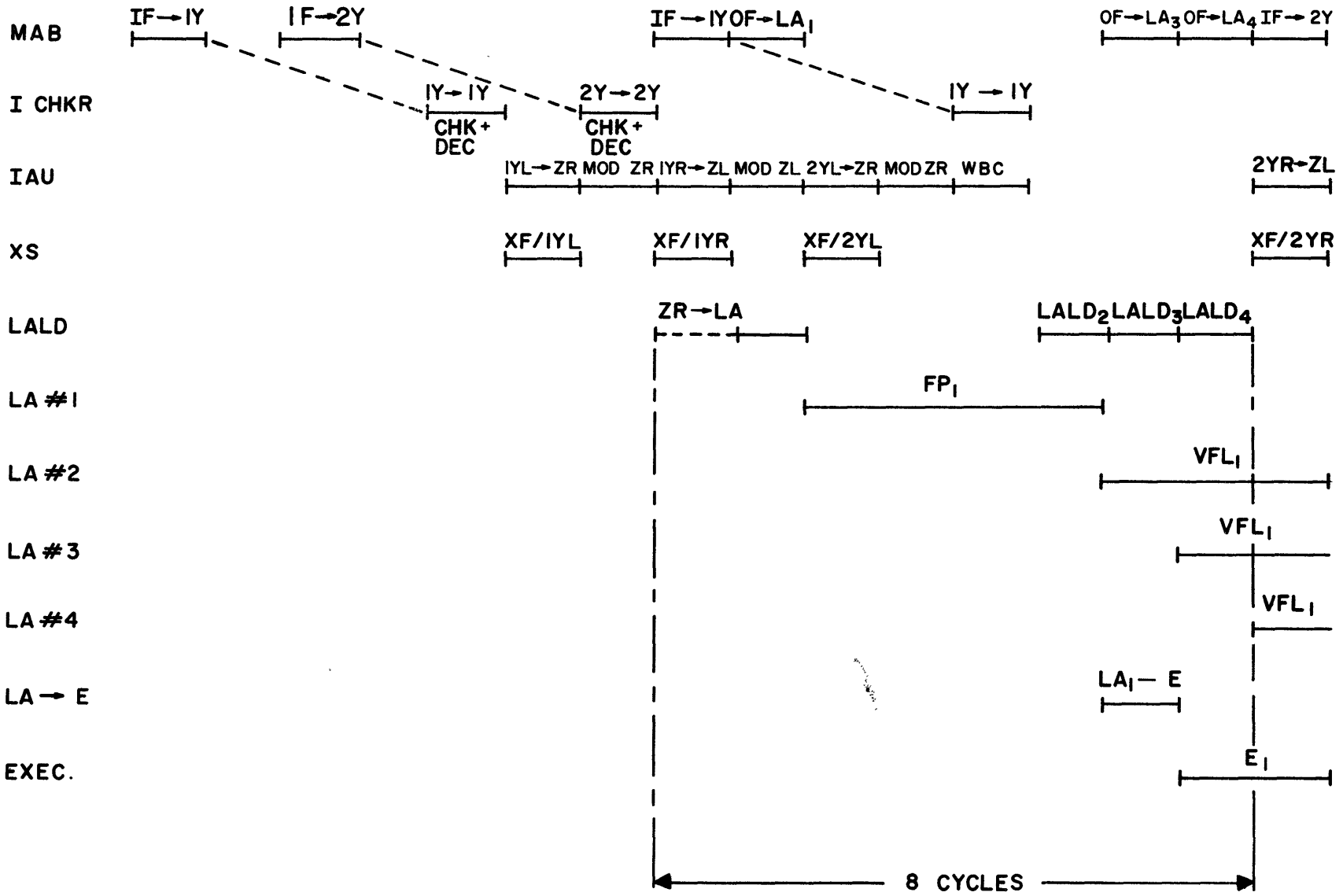


Fig. 5. Floating Point Instruction Preparation.



ASSUMPTIONS;

1. VFL INST. ACROSS MEM. WO. BOUNDS
2. NORMAL INDEX BOTH HALVES.
3. TWO OPNDS. REQD.

Fig. 6. VFL Instruction Preparation.

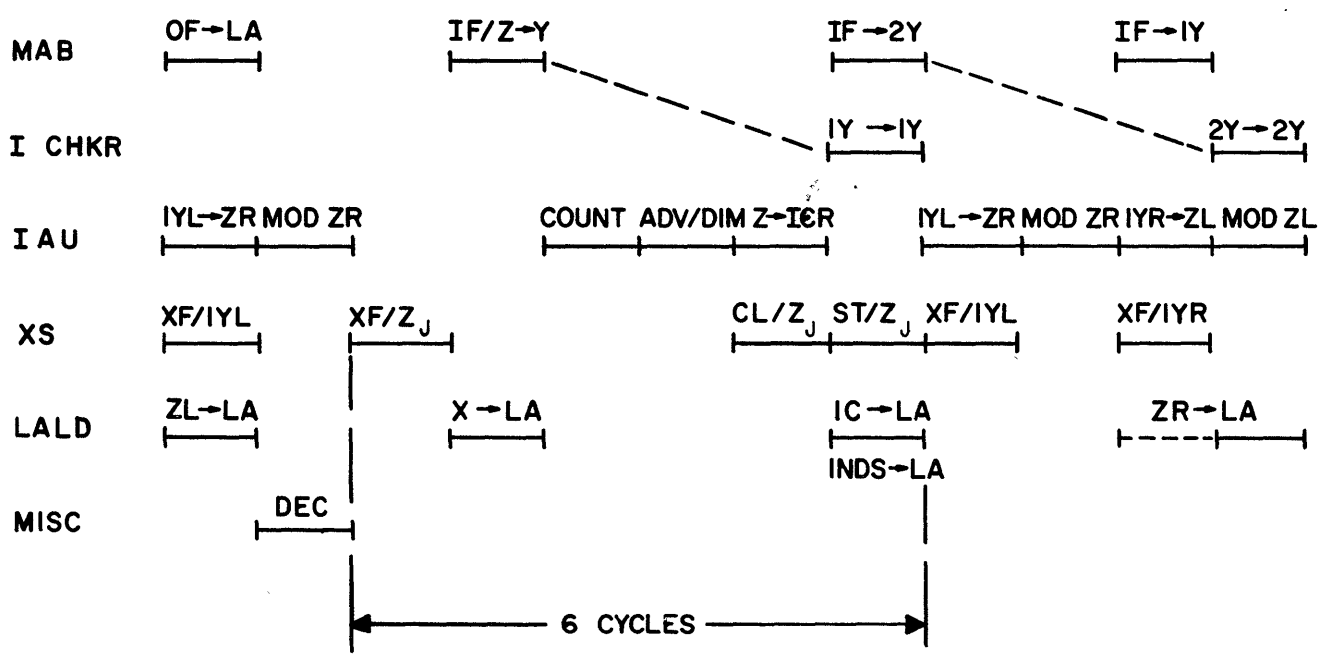
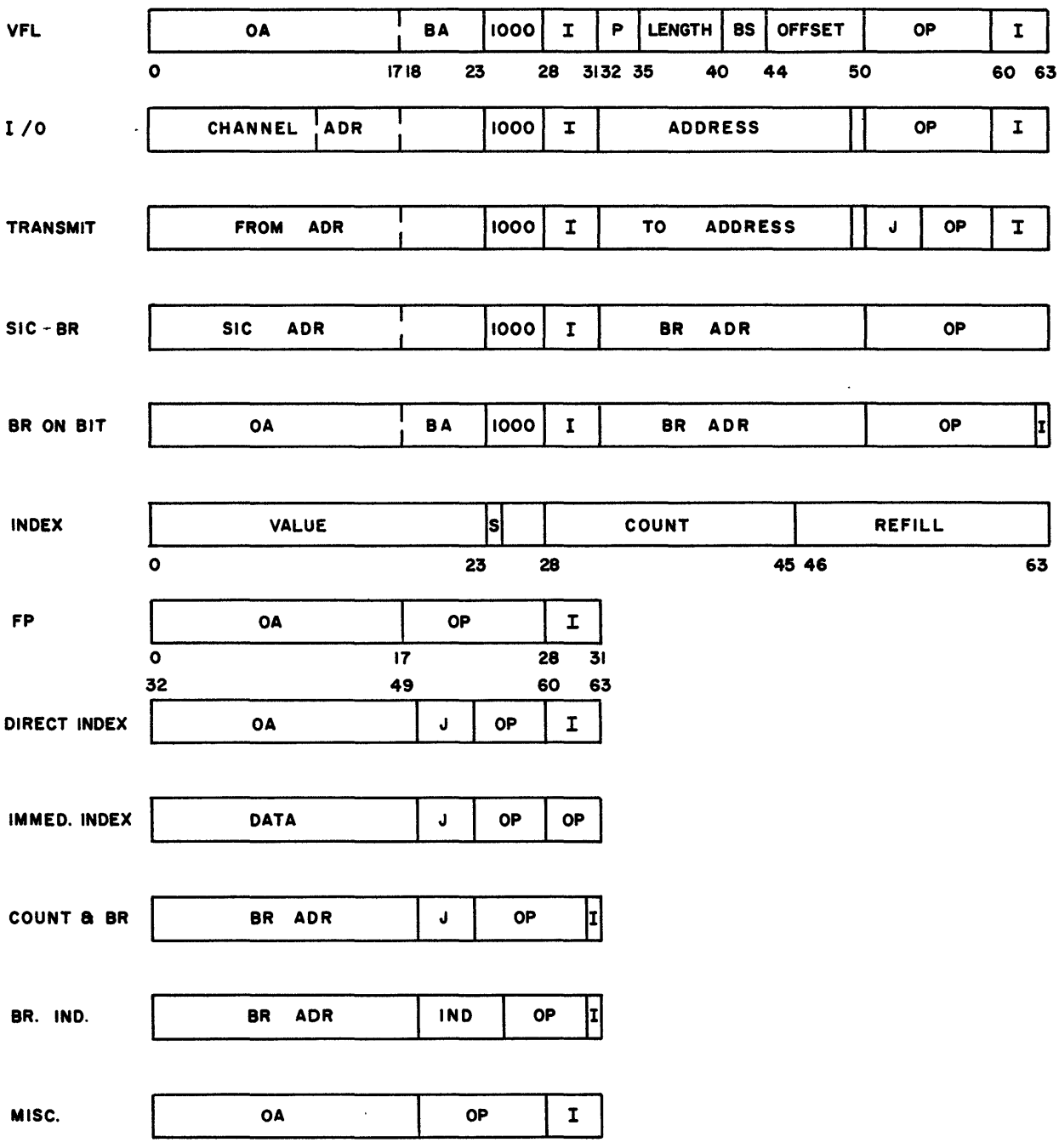


Fig. 7. Count and Branch Execution.

APPENDIX A

Instruction and Index Word Formats.



APPENDIX B
Memory Address Assignments

Location	Name	Length	Bit Address	Type
0	Zero	64	0 - 63	EM
1 P,a	Interval timer	19	0 - 18	XS
1 P,b	Time clock	36	28 - 63	XS
2 P	Interruption address	18	0 - 17	EM
3 P	Upper boundary	18	0 - 17	IR
3 P	Lower boundary	18	32 - 49	IR
3 P	Boundary control bit	1	57	IR
4	Maintenance bits	64	0 - 63	EM/MC
5 b	Channel address	7	12 - 18	IR
6	Other CPU	19	0 - 18	IR
7	Left Zeros count	7	17 - 23	IR
7	All ones count	7	44 - 50	IR
8	Left half of accumulator	64	0 - 63	IR
9	Right half of accumulator	64	0 - 63	IR
10	Accumulator sign byte	8	0 - 7	IR
11 c	Indicators	64	0 - 63	IR
12 d	Mask	64	0 - 63	IR
13	Remainder	64	0 - 63	EM
14	Factor	64	0 - 63	EM
15	Transit	64	0 - 63	EM
16-31	Index registers X0 - X15	64	0 - 63	XS
32-k	Normal external memory	64	0 - 63	EM

P — Permanently protected area of memory.

a — Read-only except for STORE VALUE, STORE COUNT, STORE REFILL, and STORE ADDRESS.

b — Read-only.

c — Bit positions 0 - 19 are read-only.

d — Bit positions 0 - 19 are always ones, and bit positions 48 - 63 are always zeros.

k — Last word address in a particular memory configuration.

IR — Internal Register

XS — Index Storage

EM — External Memory

MC — Maintenance Console

APPENDIX C

I Unit Instruction List

A. Direct Index Arithmetic

1. Load index.
2. Load value.
3. Load count.
4. Load refill.
5. Store index.
6. Store value.
7. Store count.
8. Store refill.
9. Add to value.
10. Add to value and count.
11. Compare value.
12. Compare count.
13. Rename.
14. Load value effective.
15. Store value in address.

B. Immediate Index Arithmetic

1. Load value immediate.
2. Load count immediate.
3. Load refill immediate.
4. Load value negative immediate.
5. Add immediate to value.
6. Add immediate to value and count.
7. Add immediate to value and count and refill.
8. Subtract immediate from value.
9. Subtract immediate from value and count.
10. Subtract immediate from value and count and refill.
11. Add immediate to count.
12. Subtract immediate from count.
13. Compare value immediate.
14. Compare value negative immediate.
15. Compare count immediate.
16. Load value with sum.

C. Unconditional Branching

1. Branch.
2. Branch relative.
3. Branch enabled.
4. Branch disabled.
5. Branch enabled and wait.
6. No operation.

D. Indicator Branching

Branch on Indicator

Modifiers:

a) Leave indicator.
b) Set indicator to zero.
c) Branch if off.
d) Branch if on.

E. Bit Branching

Branch on bit

Modifiers:

a) Leave bit.
b) Invert bit.
c) Set bit to zero.
d) Branch if off.
e) Branch if on.

F. Index Branching

1. Count and branch.
2. Count, branch, and refill.

Modifiers:

a) Branch if count non-zero.
b) Branch if count zero.
c) Leave value is changed.
d) Add half to value.
e) Add one to value.
f) Subtract one from value.

G. Store Instruction Counter If

H. Transmit Operations

1. Transmit
2. Swap.

Modifiers:

a) Forward
b) Backward
c) Direct count
d) Immediate count

J. Miscellaneous Operations

1. Refill.
2. Refill on count zero.
3. Execute.
4. Execute indirect and count.
5. Store zero.

THE PRINTED MOTOR: A NEW APPROACH TO INTERMITTENT AND CONTINUOUS MOTION DEVICES IN DATA PROCESSING EQUIPMENT

R. P. Burr
Circuit Research Company
33 Sea Cliff Avenue
Glen Cove, New York

Summary

The printed d-c motor is characterized by high pulse torque capability and freedom from cogging or preferred armature positions. These attributes lead to a variety of applications in data processing equipment ranging from reel and capstan drives in magnetic and paper tape transports through detenting and positioning mechanisms.

Analysis of the motor on a velocity basis yields a simple equivalent circuit which is a powerful tool for designing both the machine and its drive circuits into a specific requirement. Since there is no rotating iron in the structure and since the field is supplied by permanent magnets, the speed-torque curve of the motor is a straight line whose slope defines a "mechanical source impedance." Inertia of the proposed load appears as a capacitor in the same dimensional system. When the desired machine motion can be expressed in terms of velocity and the inertia of the load is known, the shape and magnitude of the necessary driving signal can be determined for the operating cycle.

A typical example of an application in a paper tape transport is described.

Machine Structure and Printed Armature

Most of the important electromechanical characteristics of the printed motor are shown in the exploded view of a typical small machine as given in Figure 1. Here we see the mechanical relationship of the five key elements of the device: exciting magnets, armature, field return path, brushes and bearings. In the photograph the eight pole pieces of the permanent magnet cage are visible to the right. These are polarized alternately north and south, so that there are actually four pole pairs in the structure. One should imagine that magnetic flux leaves some particular pole, travels parallel to the machine axis and traverses the armature disc into the soft iron return ring at the left. Within the return iron the flux divides and travels in both directions circumferentially until it arrives under adjacent magnetizing poles of opposite polarity. At this point it re-emerges into the air gap, passes again through the armature and thence back into the magnet. Hence, we have what is in effect a planar or axial air gap d-c motor structure in which the torque producing conductors rotate independently of the magnetizing iron, and in which, reciprocally, the uniformity and smoothness of the magnetic field is virtually unaffected by rotation of the armature. The photograph also shows the brush locations. One may observe that commutation takes place directly upon the surface of the armature disc so that every conductor is commutated in sequence. This point is of considerable importance when added to the fact that rotation of the armature does not modulate the flux density. The result is that the printed motor displays no cogging whatever: the armature has no tendency toward a preferred set of positions.

It would be difficult to imagine a d-c motor of simpler construction. The key, of course, lies in the design and fabrication of the printed armature which serves not only as the motor winding but as the commutator, or vice versa. The details of this important component need not concern us here since they have been discussed elsewhere in the literature[1]. Suffice it to say for present purposes that the armature is produced by modern printed circuit techniques from a master drawing: the conductors have a flat cross-section, are uninsulated in the conventional sense, and are carried upon a substrate of mylar, epoxy-glass, or high-alumina ceramic, as the application demands. The technique may be practiced

for machines ranging in power output up to several mechanical kilowatts and in peak torques from a few ounce-inches to several thousand pound-feet. In each case the armature may be thought of as the "annular" equivalent of a wave winding in which commutation takes place upon every active conductor.

Electro-mechanical Characteristics

In order to visualize applications for the printed motor we must first examine its electro-mechanical characteristics and attempt to construct an analogy for its performance as a circuit component. As with most physically uncomplicated structures, we shall find this approach to be both easy and gratifyingly simple.

A first step is to consider the relationship between speed and torque which may be deduced from an inspection of Figure 3, which is a schematic of the machine from an electrical point of view. Resistor r_a is defined as the armature resistance plus brush resistance plus the resistance of the source of voltage E_T, if any. Voltage e_a is the back EMF and is related to the speed S in thousands of revolutions per minute (or some other convenient unit) by a constant factor k_e. Similarly, the output torque, T, is given by $T = k_T i_a$ where k_T is another constant factor. Application of Ohm's law to the circuit yields the classical equation for an ideal shunt wound or permanent magnet field d-c motor:

$$E_T = \frac{T}{k_T} r_a + k_e S \qquad (1)$$

A slight rearrangement of this expression leads to:

$$\frac{E_T}{k_e} = T \frac{r_a}{k_e k_T} + S \qquad (2)$$

which is the equation of a straight line as shown in Figure 4 having a negative slope numerically equal to $r_a/k_e k_T$.

Before continuing, we should pause to note that no component for electrical inductance is shown in Figure 3. In point of fact, the inductance of a printed motor armature is so small as to be negligible; torque at stall is in phase with the applied voltage. Subsequently, we shall be discussing intermittent operation of the motor at frequencies of several hundred cycles per second so that this point should not be neglected.

Equivalent Circuit Development

Equation (2) and the plot of Figure 4 show clearly that the end points of any family of speed-torque curves are a function only of the motor terminal voltage, E_T. A further step may be taken by recognizing that the slope of these curves, $r_a/k_e k_T$, is a useful measure of quality in such machines. The dimensions of the slope are speed-change-per-unit-applied-torque and it may be thought of as resulting from an equivalent series mechanical resistance, which we shall designate by the symbol R_{ms}.

Upon pursuing this idea a little further, we are led to the concept of an electromechanical equivalent circuit on the basis of speed as shown in Figure 5. Here the usual voltage source is replaced by a generator of speed, S_i, delivering an "output" S_o through an internal resistance, R_{ms}. The flow of torque, T, through R_{ms} causes a speed drop as shown in Figure 4. Torque is therefore analagous to current. Nothing is lost in regarding the machine from this viewpoint, since Figure 3 and Figure 5 are identical except for a change of dimensions. Consideration of the power relationships in both circuits will demonstrate this fact.

The circuit of Figure 5 is a first approach and is adequate to describe the motor only so long as we are dealing with steady state conditions and torque loads (including friction from bearings and brushes) which are independent of speed. Practical experience with and analysis of printed motors has shown that the total loss torque to be expected must always contain a viscosity component, or a torque which is proportional to output speed, in addition to a constant value. Graphically the behavior is as shown in Figure 6. To be consistent with our previous concept of mechanical resistance we will assign the symbol R_{mD} to the slope of the viscosity or damping torque line. The damping component of torque to be expected at any speed is therefore S_o/R_{mD}.

By substituting the speed-varying torque component into equation (2) or the circuit of Figure 5, one obtains:

$$S_o = S_i - (T_F + S_o/R_{mD})R_{ms} \qquad (3)$$

which, after some algebra, becomes:

$$S_o = S_i \frac{R_{mD}}{R_{ms}+R_{mD}} - T_F \frac{R_{ms}\ R_{mD}}{R_{ms}+R_{mD}} \qquad (4)$$

whereupon we observe with the aid of Thevenin's Theorem that R_{mD} is a shunt "resistor" across the output terminals of the network, as in Figure 7.

To complete the network for transient operating conditions, which are usually of chief concern in servomechanisms, we must provide a component for the mechanical inertia of the motor and load. By inspection of the equations of motion for a printed motor, one arrives at the conclusion that mechanical inertia may be represented as a capacitor of appropriate magnitude connected in parallel with R_{mD} across the network output terminals. In particular, a possible equation of motion for the circuit of Figure 5 in response to a velocity "step", S_i, is:

$$S_o = S_i - J \frac{dS_o}{dt} \times R_{ms} \qquad (5)$$

where J denotes total system inertia. Rearranging this slightly we get

$$\frac{dS_o}{dt} + \frac{S_o}{JR_{ms}} = \frac{S_i}{JR_{ms}} \qquad (6)$$

which is solved to obtain:

$$S_o = S_i \left(1 - e^{-\frac{t}{JR_{ms}}}\right) \qquad (7)$$

In this equation the factor $J \cdot R_{ms}$ is the system mechanical time constant, τ, and is to be compared with the analagous factor $R \cdot C$ in a simple resistance capacitance network.

The complete equivalent circuit is shown in Figure 8. Two sets of consistent units are given in Table I, while specific values of the various elements for several types of printed motors are listed in Table II.

Interpretation of Equivalent Circuit

Having now devised a useful tool to aid in thinking about the motor we are in a position to draw several conclusions:

First, we note that the velocity response of the motor to changes of the input speed (i.e., applied terminal voltage) or output torque will be exactly analagous to the electrical response of the equivalent single-time-constant RC network. Conclusions on frequency response, power dissipation, source impedance and the like are equally valid for either system. This is a comfortable fact because it means that the motor is a simple element to consider in the overall design of a servomechanism.

Second, the arrangement of the circuit indicates clearly that an increase of the viscous damping component is beneficial to the system response time. The corresponding reduction of R_{mD} lowers the motor time constant, i.e., we broaden the bandwidth by loading the "capacitor." At the same time, excessive damping will result in an inefficient machine useful only at low speeds, since the speed attenuation through R_{ms} into R_{mD} ultimately becomes large.

Third, for high performance the resistance of the "generator" is extremely important and should be kept low with respect to the armature and brush resistance. Otherwise the effective value of R_{ms} will rise above the theoretically attainable value and the time-constant will increase.

Fourth, the performance of the motor as a transducer is critically dependent upon flux density in the air-gap. Both k_e and k_T vary directly with field strength, so that the value of both R_{ms} and the time-constant vary inversely as the square of the field.

Fifth, the transition between the velocity input to the network and the actual driving voltage is given by the factor k_e, since $E_T = k_e S_i$. This is to say that once the desired input velocity has been determined the corresponding driving voltage is in direct proportion.

Application

Two general types of servos are of interest in the field of data processing equipment. The first covers the range of motor applications in velocity servos for magnetic tape reel and capstan drives, analog multiplying devices and similar continuous motion applications. The second area contemplates intermittent operation of the machine as in paper tape drives, card handling apparatus, printers and so on. In this field, the desired result usually involves the rapid and precise acceleration and deceleration of some medium from one position to another in a manner which may not be repetitive. Control is usually provided by some form of pulse position servo, the signals to the motor being simply "ON" or "OFF". The equivalent circuit is particularly useful for this type of operation since the velocity behavior in either of the two states can be easily predicted.

Figure 9 shows a typical arrangement for the step-by-step advance of perforated paper tape. Let us assume that it is desired to eliminate all brakes and clutches from the transport and to accomplish movement of the tape simply by starting and stopping the motor,--which is directly coupled to the tape drive capstan. Further, we wish that slewing operation with stop on a character be available without modification to the system. To achieve these conditions, the following sequence of events can be supposed:

1. Start: The motor is at rest with the tape in registry over the reading aperture. A clock or command pulse is received, switching the electronics and the power transistor controlling the motor into the conducting state.

2. Run: The motor velocity rises rapidly to its terminal value (or very nearly) before the next sprocket hole appears in the reading aperture. (This condition guarantees a stop will occur in registry independent of the tape travel while slewing.)

3. Stop: A stop pulse is received from the tape, switching off the electronics and the motor. The motor velocity decays to zero in the interval between the arrival of the sprocket hole at the edge of the reading aperture and the movement of the hole to a position of good registry.

In order to arrive at the desired velocities in the network, we must calculate from the various times, velocities and displacements implied by the specifications. One commences the design by adjusting the diameter (and therefore the inertia) of the capstan drum so as to maximize linear acceleration at the drum periphery. Next, the total permissible time for the motion may be calculated from the maximum stepping speed required; it might be, for example, 5 milliseconds for standard one-inch paper tape. For the time period from Start to Stop as previously described, one must assume that about 2.5 time constants elapse, so as to be sure that the motor is near terminal speed when the Stop signal is received. Further, it is reasonable to assume (by calculation from the equivalent circuit) that the motor may be effectively stopped in about one time constant with a small amount of reverse pulsing. Therefore we conclude that the entire motion time must occupy about 3.5 time constants, so that one time constant must equal about 5/3.5 or 1.4 milliseconds. Finally, we can estimate the various displacement components of the cycle accurately, since these are given by the drum diameter and the dimensions of the tape. From the information on motion time, drum diameter, running displacement and time constant, one may calculate and plot a curve of the motor output velocity which must be delivered by the equivalent circuit. A typical result is sketched in Figure 10. Note that a reverse pulse is shown as beginning at Stop. The negative voltage pulse applied to the motor terminals during this interval is such that it would result in a complete reversal of the machine along the dotted line extending into the negative velocity region if sufficiently prolonged.

The corresponding velocity (or

terminal voltage) input to the network is shown in Figure 11. The forward or "run" velocity is effectively that obtaining at the instant of Stop in Figure 10, since we assume that terminal velocity has been achieved during the "run" period. The reverse velocity amplitude must be consistent with the assumption of a complete stop in about one time constant. This is the same thing as saying that the reverse velocity asymptote shown in Figure 10 must be

$$1 - \frac{1}{1 - 1/e}$$

times the forward velocity. The pulse duration is set close to the expected stopping time, but is not critical since an error in timing when the motor velocity is near zero results in only a small displacement.

Reduction to practice with such a design procedure is fundamentally a series of successive approximations: the designer is bound by limitations of the motor itself in terms of the realizable values of R_{ms} and R_{mD} which may be employed to get a usable result. Further, the effect of friction torque is always beneficial to intermittent motions if it is not excessive and is difficult to assess until the physical apparatus is tested: friction is easily compensated by raising the forward velocity and assists in a rapid stop since the friction torque "current" discharges the inertia "capacitor." In general terms, the equivalent circuit approach is valuable since it provides a point of departure and as a means for reconciling unexpected phenomena which are so often experienced in matters of this sort.

Developmental Tape Reader

The circuit parameters and schematic driving arrangement for a newly developed high performance printed motor in a developmental paper tape reader are shown in Figure 12. Note that the damping resistor, R_{mD}, which would normally be several times R_{ms} (Table II), is adjusted to be half the series resistance. In practice this is accomplished by laminating the armature with a thin disc of soft aluminum. The tape capstan drum and other mechanical parts of the machine are carefully adjusted to achieve low inertia.

Operating tests on this reader transport show that a mechanical time constant of about one millisecond has been achieved. The motor will execute the complete cycle of start, accelerate, run, shut-off, decelerate and stop at a rate of 250 lines per second for standard one-inch paper tape. Stop-on-a-character is automatically obtained for slewing operation while the reader will advance blocks of tape ten lines long at a rate of 25 per second.

Conclusion

The printed motor is an electrically simple device, providing smooth torque from a low inertia mechanical structure. Linearity of the speed-torque curves allows the construction of a simple equivalent circuit which accurately represents the transient performance of the machine in continuous and intermittent motion servos. The feasibility of a proposed application may be studied and a design formulated by reconciliation of the desired velocity behavior with the machine parameters.

1 Henry-Baudot, J. and Burr, R. P., "Printed Circuit DC Motors for Electronic and Instrument Applications," IRE National Convention Record, Part 9, March 1959 and Burr, R. P., "Printed Circuit Motors" presented at 1959 AIEE Machine Tool Conference, Cleveland, Ohio, October 20, 1959.

Table I

Consistent Units

Equivalent Motor Speed Network

A. Speed is measured in 1000 rpm, denoted by kpm.
Armature resistance, r_a, in ohms.
Torque per ampere, k_T, in in-oz/ampere.
Back EMF per kpm, k_e, in volts/kpm.
Mechanical resistance, R_{ms}, R_{mD} in kpm/in-oz.
Inertia, J, in 105 x in-oz-sec^2.
Terminal voltage, E_T, in volts.

Note: The multiplying factor 105 for J arises from a conversion between radians per second and kpm.

B. Speed is measured in radians per second, rad/sec.
Armature resistance, r_a, in ohms.
Torque per ampere, k_T, in gr-cm/ampere.
Back EMF per kpm, k_e, in volts/rad/sec.
Mechanical resistance, R_{ms}, R_{mD} in rad/sec/gr-cm.
Inertia, J, in gr-cm-sec^2.
Terminal voltage, E_T, in volts.

Table II

Motor	R_{ms} kpm/in-oz	R_{mD} kpm/in-oz	τ milli-seconds	Cont. Torque in-oz	Peak Torque in-oz	Case Diameter inches
PM 368	94.5×10^{-3}	842×10^{-3}	34	12	150	4.250
PM 368 A	94.5×10^{-3}	13.2×10^{-3}	5	12	150	4.25
PM 488	23.5×10^{-3}	400×10^{-3}	41	42.5	375	5.625
PM 668	2.5×10^{-3}	106.6×10^{-3}	24	140	1175	7.125
PM 1028	242×10^{-3}	12.13×10^{-3}	23	1000	8400	11.0
PM 368 HF	15×10^{-3}	7×10^{-3}	1.5	30	375	5.625

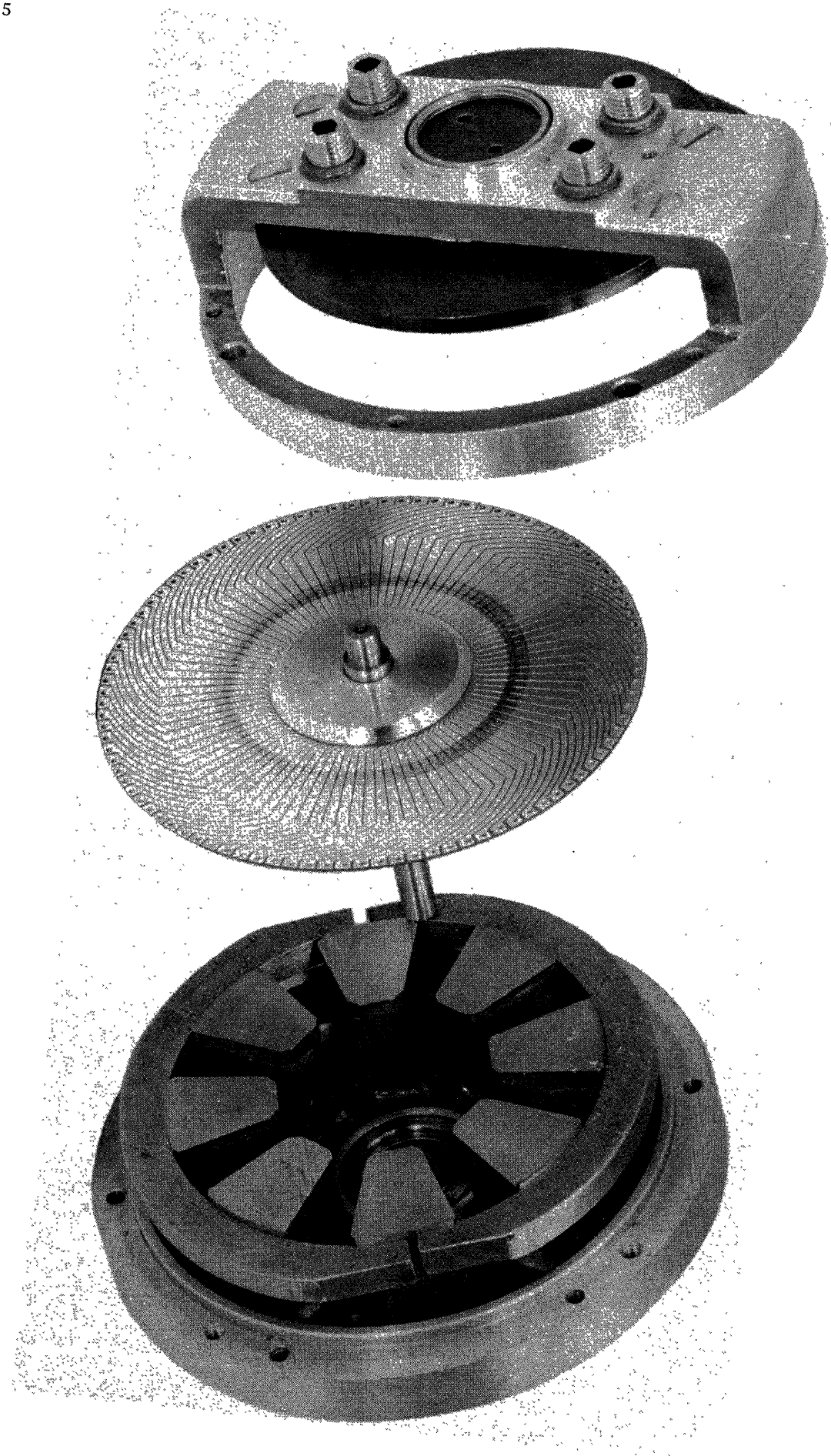


Fig. 1. Printed Motor, Exploded View

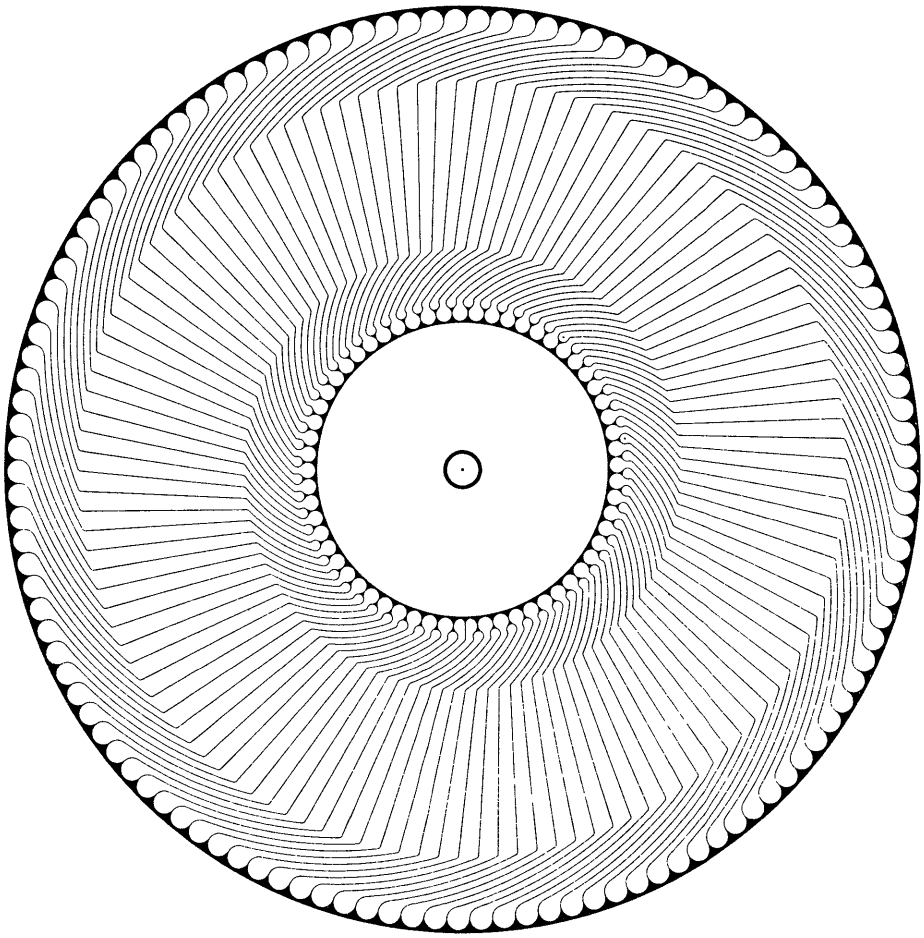


Fig. 2. Design of Printed Circuit Armature

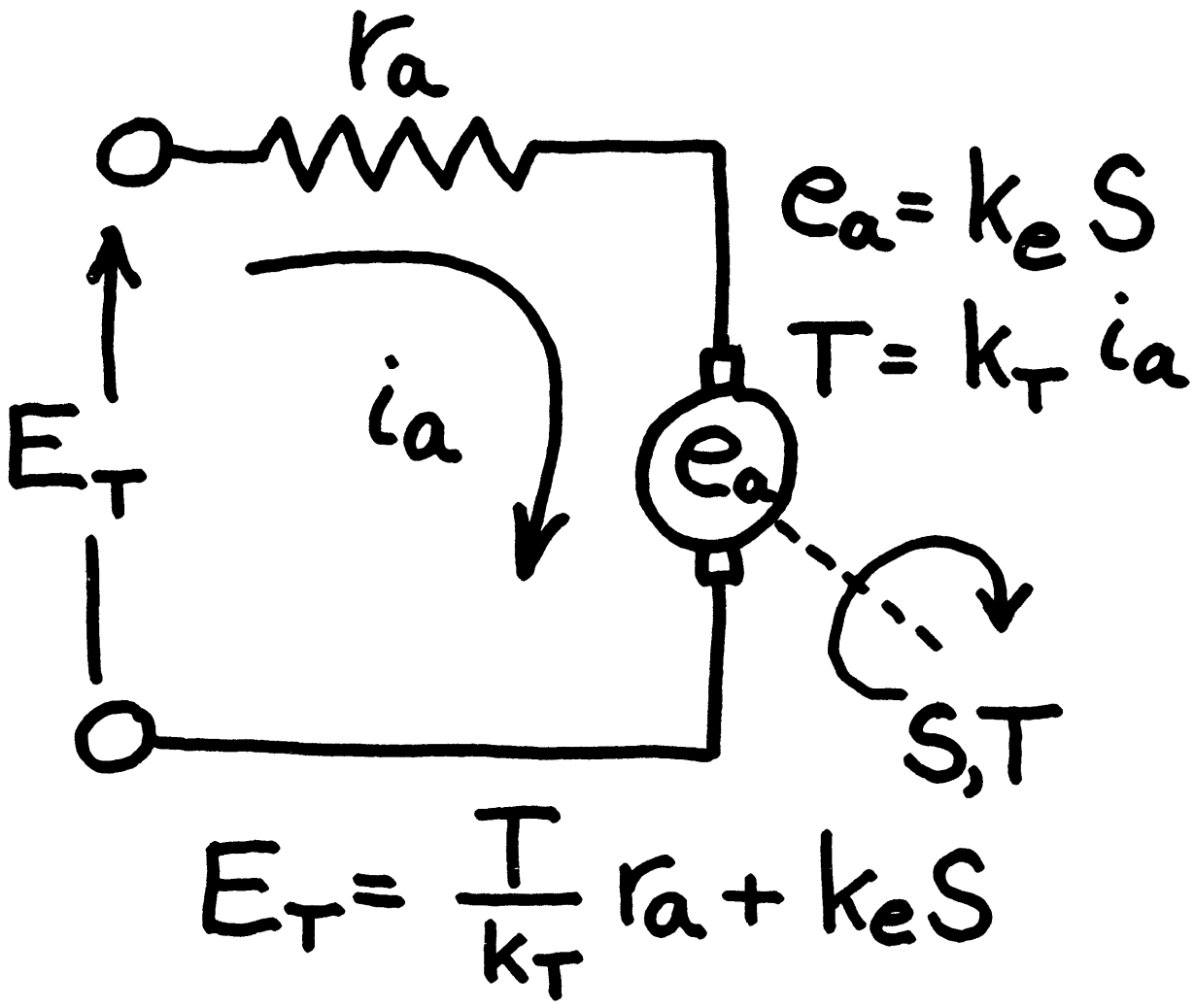


Fig. 3. Electrical Schematic of Ideal Shunt-Wound or Permanent Magnet D-C Motor.

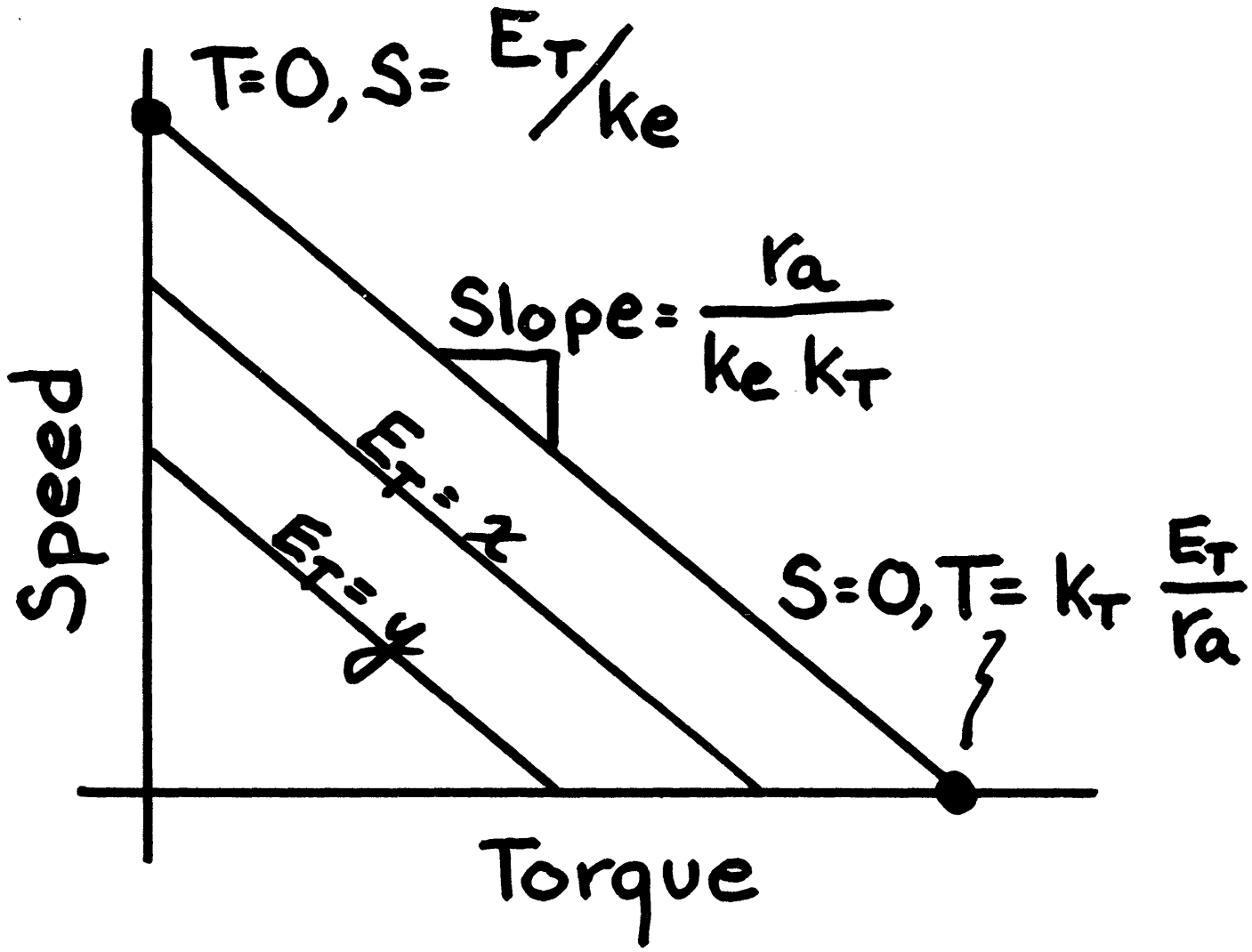


Fig. 4. Printed Motor Speed Torque Curves

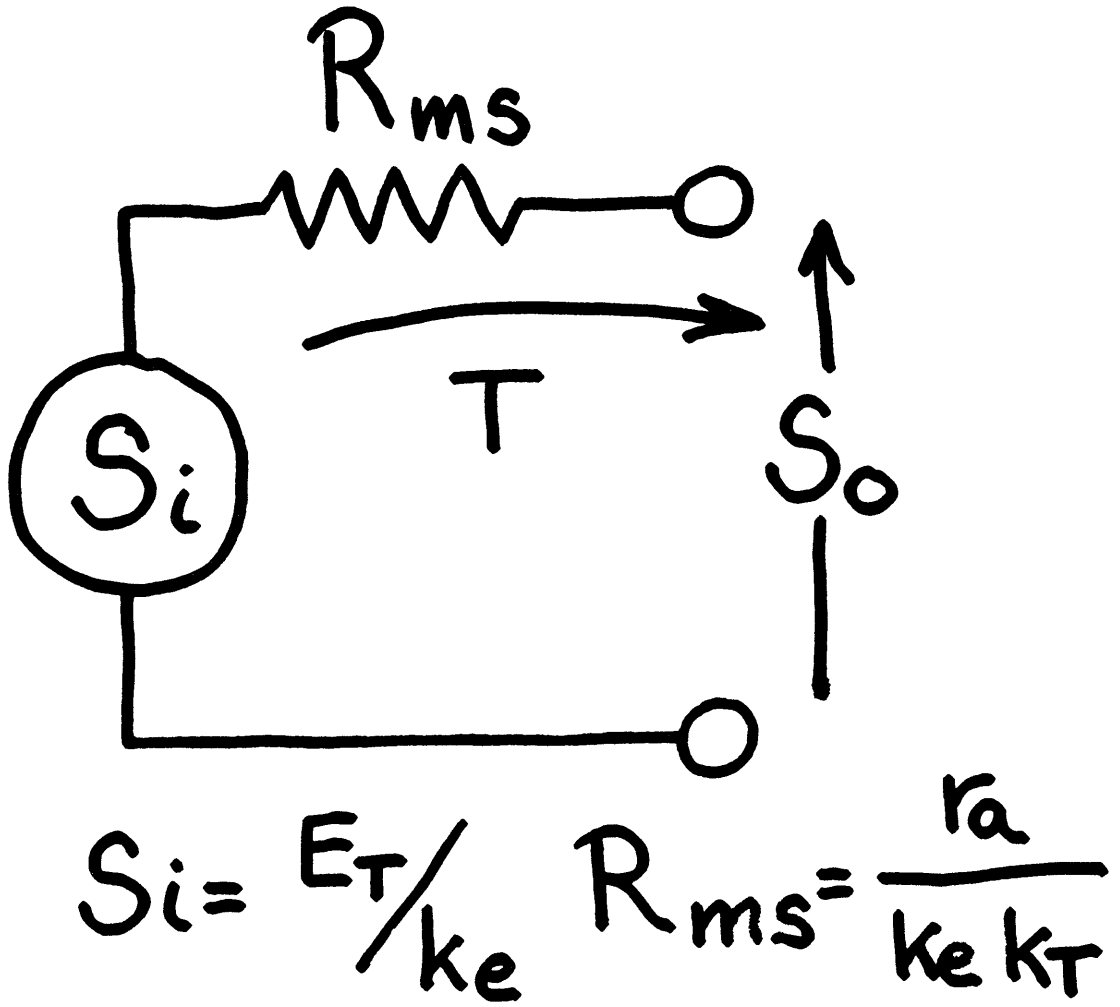


Fig. 5. Elementary Mechanical Impedance Concept

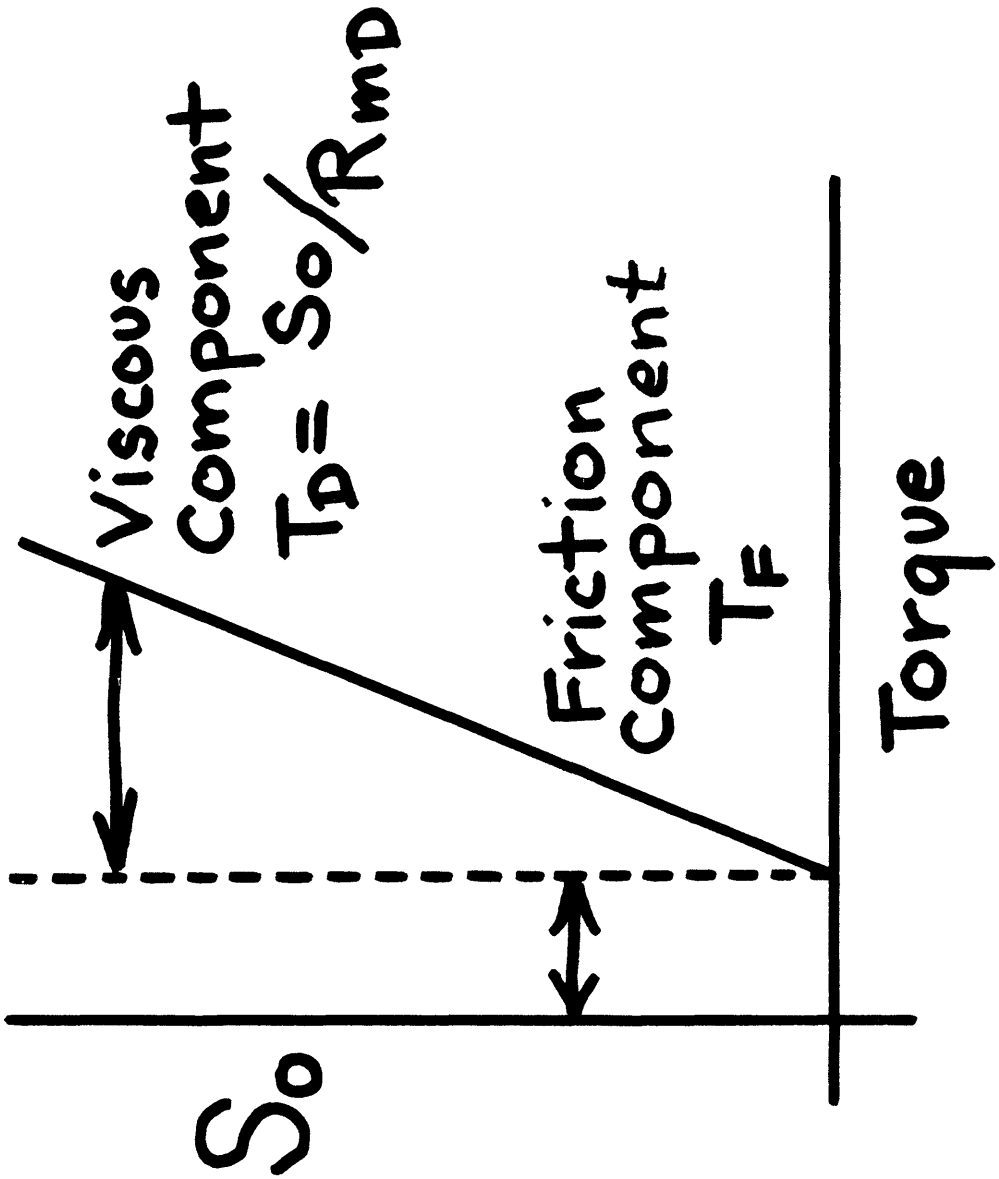


Fig. 6. Composition of Loss Torques in a Printed D-C Motor

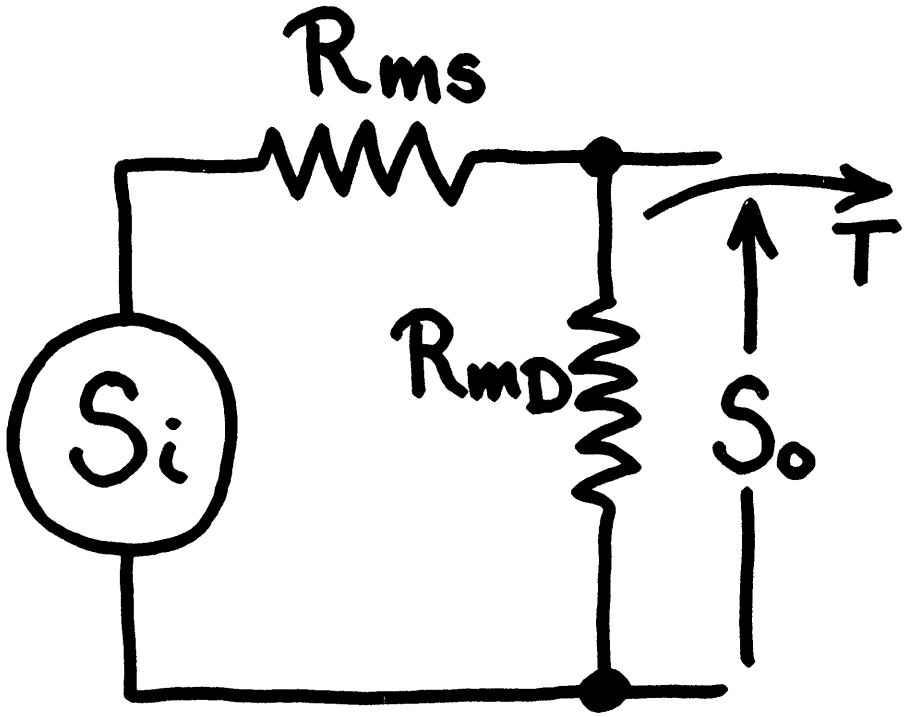


Fig. 7. Location of Mechanical "Viscosity" Resistance

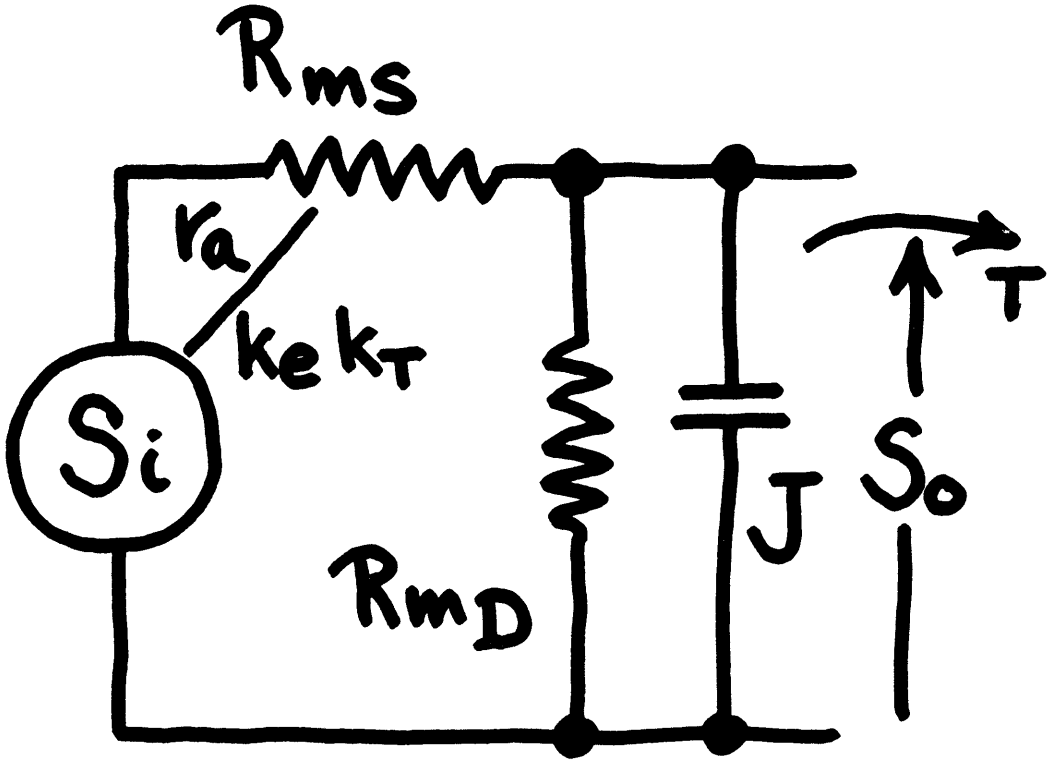


Fig. 8. Complete Equivalent Speed Network Including Inertia Capacitor

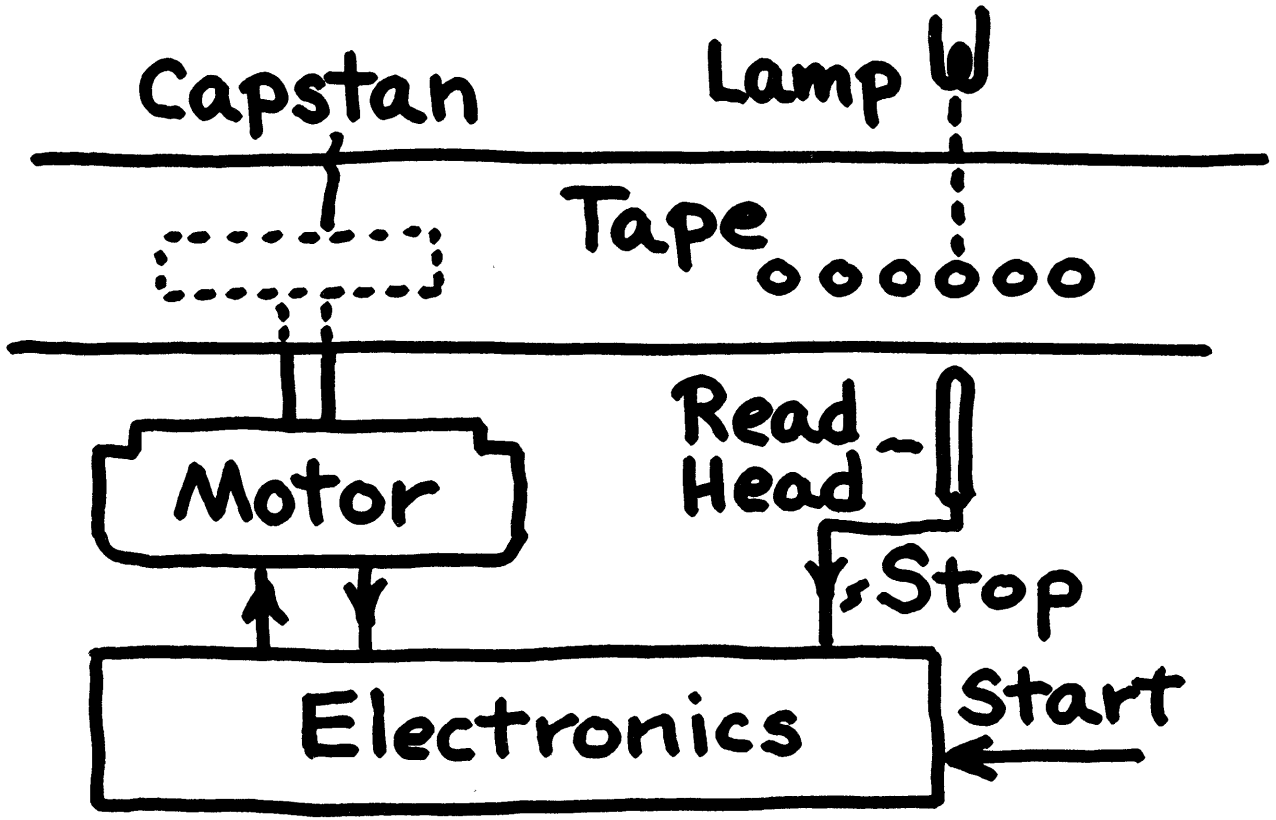


Fig. 9. Operating Principle of Step-by-Step Paper Tape Reader

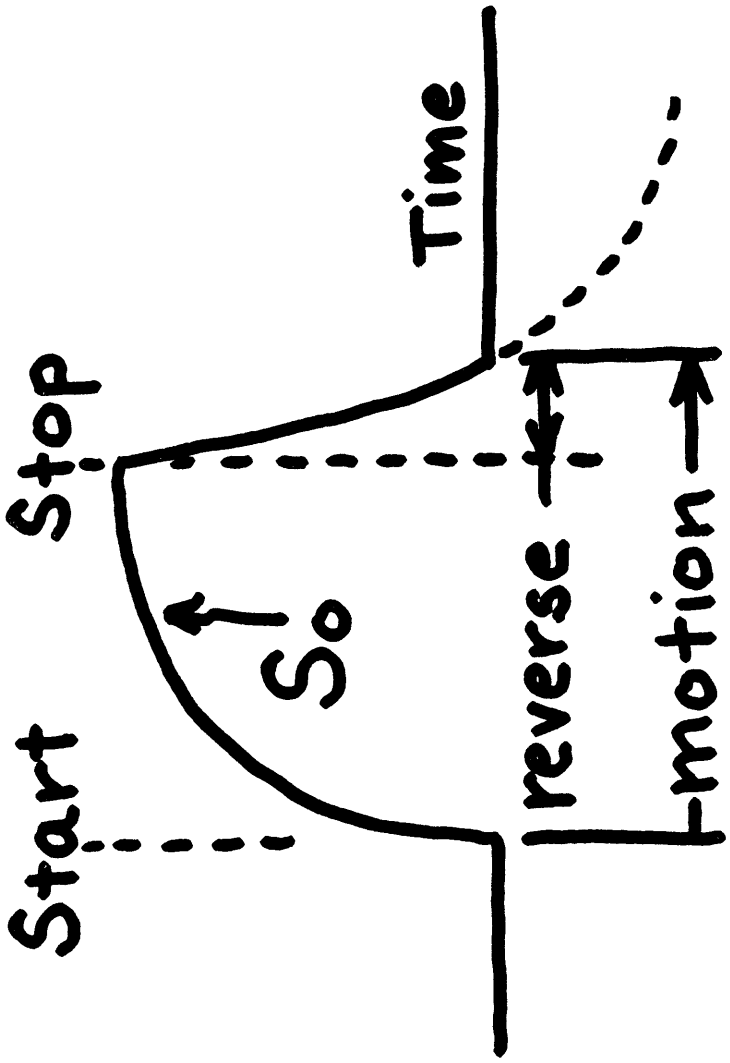


Fig. 10. Output Velocity Waveform Involved in Paper Tape Transport

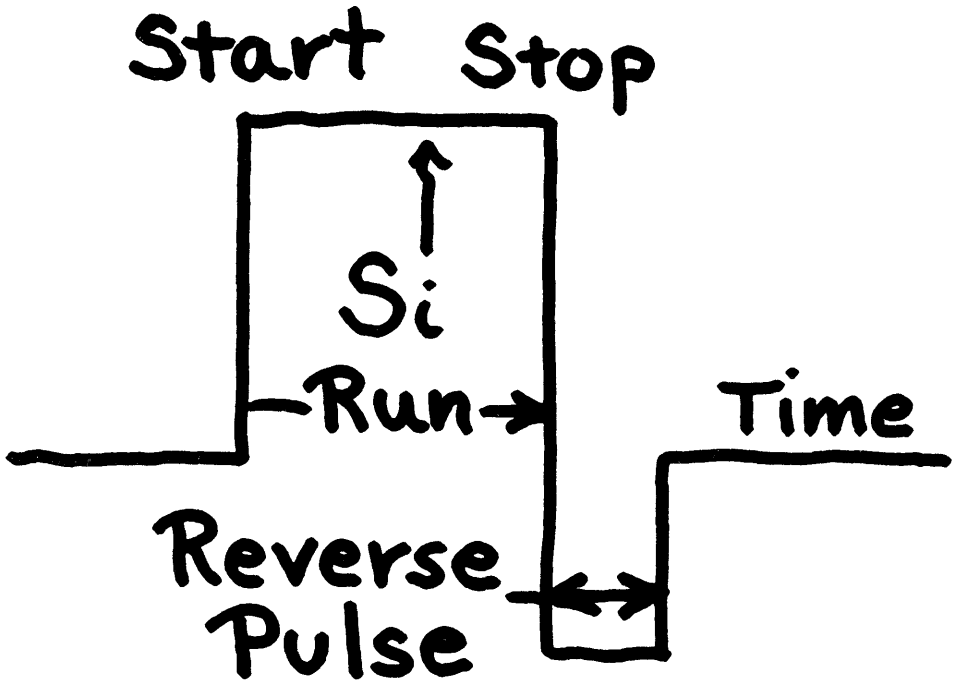


Fig. 11. Input Velocity Waveform for Paper Tape Transport

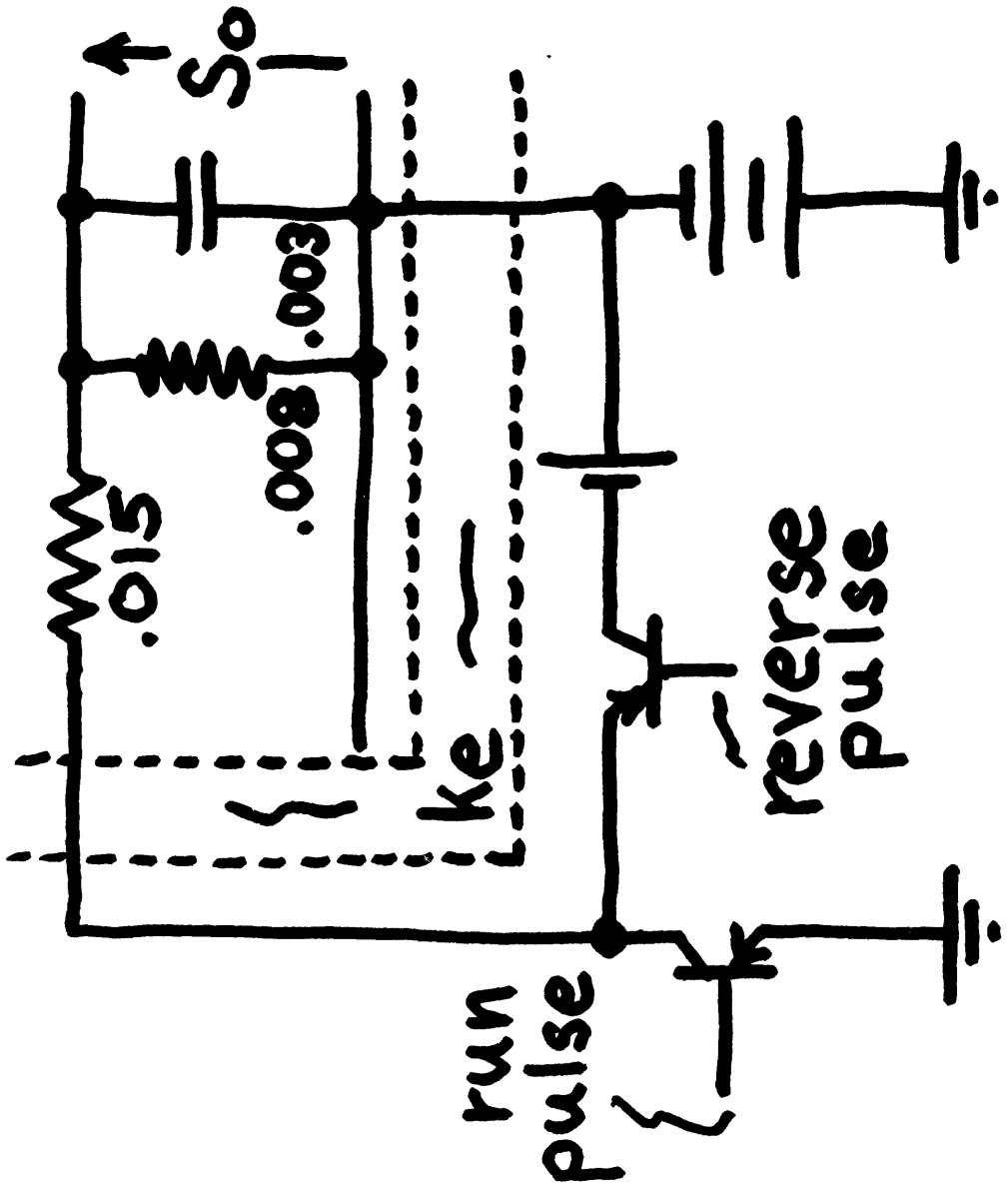


Fig. 12. Schematic Diagram of High-Performance

INDEX BY AUTHORS

(Continued)

INDEX BY AUTHORS, continued

(Continued)

INDEX BY AUTHORS, continued

LIST OF EXHIBITORS

American Telephone & Telegraph Co.	New York, New York
Amp, Inc.	Harrisburg, Pennsylvania
Ampex Data Products Co.	Redwood, City, California
Anelex Corp.	Boston, Massachusetts
Audio Devices, Inc.	New York, New York
Automatic Electric Co.	Northlake, Illinois
Autonetics	Downey, California
The Bendix Corp.	Los Angeles, California
Bryant Computer Products	Walled Lake, Michigan
Burroughs Corp.	Detroit, Michigan
CBS Laboratories	Stamford, Connecticut
C-E-I-R, Inc.	Arlington, Virginia
C & K Components, Inc.	Newton, Massachusetts
C.P. Clare & Co.	Chicago, Illinois
C.P. Clare Transistor Corp.	Glen Head, L.I., New York
Computer Control Co., Inc.	Framingham, Massachusetts
Consolidated Electrodynamics Corp.	Pasadena, California
Control Data Corp.	Minneapolis, Minnesota
Di/An Controls, Inc.	Boston, Massachusetts
Datamation	New York, New York
Digital Equipment Corp.	Maynard, Massachusetts
The Digitan Co.	Pasadena, California
Digitronics Corp.	Albertson, L.I., New York
Dynacor, Inc.	Rockville, Maryland
Elco Corp.	Philadelphia, Pennsylvania
Electronic Associates, Inc.	Long Branch, New Jersey
Engineered Electronics Co.	Santa Ana, California
Epsco, Inc.	Cambridge, Massachusetts
Fairchild Semiconductor Corp.	Mountain View, California
Ferranti Electric	Hempstead, New York
GPS Instrument Co., Inc.	Newton, Massachusetts
General Ceramics	Keasbey, New Jersey
General Electric Co.	Johnson City, New York
Genesys	Los Angeles, California
Kenneth E. Hughes Co., Inc.	Union City, New Jersey
Hughes Semiconductor Division.	Newport Beach, California
International Business Machines Corp.	New York, New York
Laboratory For Electronics, Inc.	Boston, Massachusetts
Lenkurt Electric Co., Inc.	San Carlos, California
Librascope, Inc.	Glendale, California
Micro Switch	Freeport, Illinois
Mnemotron Corp.	Orangeburg, New York
F.L. Moseley Co.	Pasadena, California
The National Cash Register Co.	Dayton, Ohio
Navigation Computer Corp.	Philadelphia, Pennsylvania
Packard Bell Computer Corp.	Los Angeles, California
Philco Corp.	Philadelphia, Pennsylvania
Photocircuits Corp.	Glen Cove, New York
Potter Instrument Co., Inc.	Plainview, New York
Radio Corporation of America	Camden and Somerville, New Jersey
Ramo-Wooldridge	Canoga Park, California
Reeves Soundcraft Corp.	Danbury, Connecticut
Remington Rand Univac	New York, New York

LIST OF EXHIBITORS *(Cont.)*

Reese Engineering, Inc.	Philadelphia, Pennsylvania
Royal McBee Corp.	Port Chester, New York
Soroban Engineering, Inc.	Melbourne, Florida
Sprague Electric Co.	North Adams, Massachusetts
Stromberg-Carlson Co.	San Diego, California
Sylvania Electric Systems	Waltham, Massachusetts
Tally Register Corp.	Seattle, Washington
Telemeter Magnetics, Inc.	Culver City, California
Texas Instruments, Inc.	Dallas, Texas
Union Switch & Signal	Pittsburgh, Pennsylvania
Uptime Corporation	Denver, Colorado
John Wiley & Sons, Inc.	New York, New York