PROGRAMMING
FOR THE
UNIVAC FAC-TRONIC SYSTEM

January 1953

PRICE $18.50

PREFACE

A solution to a mathematical or industrial problem resulting from an application of the UNIVAC Fac-tronic System is evolved through four broad stages, namely:

(1) Problem analysis

(2) Programming or encoding

(3) Running the problem on the computer

(4) Interpretation and proper use of results

This manual is primarily concerned with the processes of programming and such other concepts which are necessary to a coordinated study of this subject. Hence, the chapters which follow will include some references to analysis, planning and actual computer operation.

Although the material herein presented is basic and complete, it should be clear that this manual is not intended to be a comprehensive study of the subject. It should be especially evident that no attempt has been made to include a discussion of electronics which would lead to an understanding of the design of UNIVAC and to an understanding of a complete logical functioning of its component parts. However, for those who are interested in, and prepared to comprehend the fundamental operation of UNIVAC, Chapter 9 offers opportunity for an elementary understanding of these concepts. Chapter 9 is independent of the other chapters of the Manual and may be read concurrently with them. It is planned that a later treatise, Manual 2, will develop these notions further and will also include many of the more advanced techniques of programming and analysis essential to the student requiring a total understanding of the subject.

This text is a first revision of the Programming Manual dated March 23, 1951, and an attempt has been made to construct, as nearly as possible, a self-study treatise. To this end a number of accepted pedagogical techniques have been introduced. For example, many well graded illustrative problems will be found throughout the text followed by practice exercises for the reader. In addition, some review

practice exercises are included in the Appendix. The device of providing abundant repetition of basic concepts has been deliberate in an attempt to foster good learning. The preliminary discussions found in the first sections of most chapters will serve to coordinate the material in previous chapters with current developments.

Particular emphasis has been placed in the instructions to the UNIVAC. In addition to "blocking in" these instructions for emphasis, they are

(1) discussed in the body of the text

(2) assembled and repeated in the Appendix with additional pertinent information.

(3) summarized on one page and located, for convenience, in a pocket in the back of the manual.

Finally, two charts will also be found in the pocket located in the back of the manual. These charts are to be used in connection with the discussion of Chapter 9 and it will be useful to follow the descriptions in this chapter with the "loose" charts available for reference.

January 1953

Programming for the UNIVAC FAC-TRONIC SYSTEM

CONTENTS

Chapter I

Introduction to the UNIVAC Fac-tronic System

SEC. I. HISTORICAL DEVELOPMENT

Around the clock, day after day, UNIVAC Fac-tronic Systems are now being used to process information and carry out billions of complex operations. New standards of reliability and accuracy have been set by the self-checking UNIVAC equipment, and users have confirmed that their overall costs for obtaining these dependable results are significantly lower than those which would have been incurred by use of alternative devices. The UNIVAC Systems already in operation have been tried out on an increasing variety of problems, demonstrating anew for each different problem that the equipment deserves the name, Universal Automatic Computer, from which the word UNIVAC was formed. Examples range from the most complicated mathematical equations to systems of accounting and inventory control, with automatic writing of purchase orders when stocks are low. As a comprehensive tool, its ability to handle and process information efficiently and at low cost can be demonstrated both for involved scientific calculations and for all of the manifold tasks which any large business enterprise finds necessary to its operation and administration.

As the application of the UNIVAC System to all of these diverse problems continues to grow, the need for personnel trained in the use of computers grows rapidly. Large business organizations have long had methods departments or specialists in business systems whose task it was to survey their operations and investigate possible improvements. The advent of the UNIVAC System has now created a new profession, that of translating the analysis of a business system into a practical scheme for automatically producing the desired results from the available raw input information. This significant development has its counterpart in the Scientific world as well. A new kind of mathematics is being born; no longer are we content with crude approximations which were once accepted because "the more exact equations are too difficult to solve". Both design engineers and theoretical physicists are finding ways to compute the numbers they want to the accuracy they want, and in doing so they are building a new kind of mathematics--which will ultimately have as profound an influence on the Queen of the Sciences as did the

invention of the calculus.

This programming manual is but a first step toward making available the basic information needed by those who wish to learn how to control the UNIVAC System and make it do their bidding. This manual is concerned only with the nature and use of the UNIVAC Instruction Code and the general method of using "flow charts" to symbolize and analyze any systematic sequence of operations. The examples have been chosen to illustrate common techniques. This is an introduction, not an exhaustive treatise or complete handbook. Although this manual has already been revised several times, it is to be expected that some readers will be able to suggest other modes of presentation which they believe more effective. It is hoped that we may have the benefit of such suggestions for the improvement of future editions.

Before describing the main components of the UNIVAC System, the historical background of computer development will be sketched. Over one hundred years ago Charles Babbage, an English mathematician, worked hard and long over his "computing engine" which, using mechanical parts, embodied all of the versatility and generality to make it a truly general purpose computer. His ideas were excellent, but the materials and techniques available in his day were not suited to the translation of his ideas into an operating mechanism. Even had the mechanism been possible, it would not have been able to justify its cost, for its operating rate would have been slow.

Babbage's invention was forgotten until Professor Howard Aiken at Harvard University designed Mark I, the first large-scale automatic computer ever to be put in operation. This device, first put in operation in 1944 was constructed by I.B.M. using many of their standard punched card electromagnetic components. It has been operated 364 days per year ever since. Subsequently, Professor Aiken designed and built Mark II and Mark III for the Naval Ordnance Proving Ground at Dahlgren, Virginia, where these two machines are now operating. Mark II uses relays, and punched paper tapes; Mark III has a large storage capacity magnetic drums, electronic and magnetic arithmetic circuits, and paper magnetic tape for input-

output. Mark I and Mark II contain no electronic circuits, but vacuum tubes were used in Mark III.

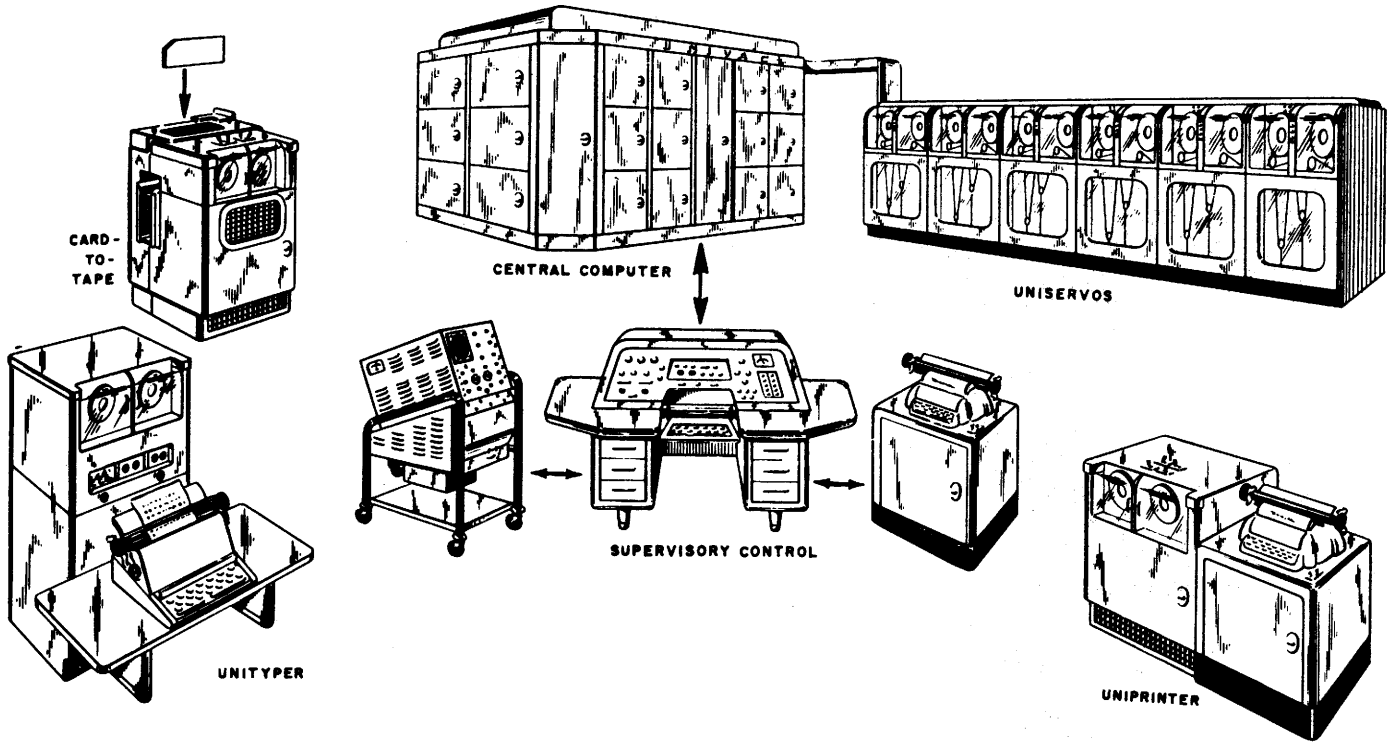
The development of electronic large-scale computers has a quite separate and independent history. In attempting to carry out statistical calculations on large volumes of weather data, Dr. John W. Mauchly became convinced that the ultimate solution would be to adapt electronic techniques to high-speed automatic computation. His initial experiments in this direction were on a small scale, but in 1943, the war-time requirements of Army Ordnance brought about a computer development contract between the Government and the University of Pennsylvania, based on a 1942 proposal by Dr. Mauchly. At this time, Mr. J. Presper Eckert, Jr. became Chief Project Engineer, and he and Dr. Mauchly together outlined a general-purpose high-speed digital computer which (except for input-output facilities) was entirely electronic. This computer, completed in December 1945 and announced early in 1946, was known as the ENIAC (Electronic Numerical Integrator and Computer). With a staff of only 12 engineers assisting them, Dr. Mauchly and Mr. Eckert were able in the short span of two and one-half years to convert their pencil-and-paper ideas into a working ensemble of almost 20,000 vacuum tubes. The ENIAC, subsequently moved to Aberdeen Proving Ground, is today operating around the clock with high efficiency and dependability.

Before the completion of the ENIAC, Dr. Mauchly and Mr. Eckert were ready with plans for a more powerful computer which, by the use of newly invented principles, could be made with far fewer tubes. Their plans were described in a classified report to Army Ordnance in September 1945, by which time the new design was known as the EDVAC (Electronic Discrete Variable Computer). Characteristic of the new design was the use of sound waves in tubes of mercury for the storage of large numbers of digits in immediate readiness for computer use. A further innovation was the decision to store all computer control instructions in these same mercury tanks, so that operations on the instructions were just as easy as operations on the data. The use of magnetic tapes for reading in the instructions and data, and for recording intermediate or final results, was also proposed.

At that time, digital recording on magnetic tapes was a novelty, and even the now-familiar recording of sound on magnetic tapes was in its infancy. One further major difference between the EDVAC and the ENIAC may be noted: ENIAC did many operations in parallel, while EDVAC was to be strictly a serial computer, doing only one operation at a time, but achieving its speed by stepping up the basic "pulse rate" from 100,000 per second in the ENIAC to at least one million per second in the EDVAC.

In 1946, Mr. Eckert and Dr. Mauchly resigned from the University and set up their own company, which began as a partnership under the name "Electronic Control Company". With a staff of only a few engineers they began, under contract with the National Bureau of Standards, to develop the components and logical plans for an electronic computer which would handle alphabetic as well as numeric data, and which would have a versatility and flexibility far beyond that usually implied by the term "computer". Although perfection of the mercury tank method of storing information was by no means easy, probably the most difficult but at the same time most necessary development effort was that required to achieve a workable and dependable high-speed magnetic tape unit for input-output use. Their success in so doing is now a matter of record, and the ready access which the UNIVAC System provides to an exceedingly large amount of information on tape is an important factor contributing to the versatile performance of that system.

SEC. 2. THE COMPONENT PARTS OF UNIVAC



The name "UNIVAC" refers to an assemblage of equipment which includes a computing unit and several auxiliary devices to provide a communication train between the computer and the human inquirer. Information is represented in the central computer by a train of electrical or acoustic pulses. Three auxiliary devices, UNITYPER, UNISERVOs, and UNIPRINTER are used to translate information between the printed page and the computer's language medium.

The UNITYPER, which contains a keyboard similar to a typewriter keyboard, converts data to a predetermined pattern of pulses impressed on magnetic tape. Each keystroke records the pulse pattern corresponding to that character on magnetic tape. Typing errors which the typist has sensed can be corrected by backspacing and retyping. The tape is automatically erased as new information is impressed over the erroneous characters.

The UNISERVOs contain magnetic reading and recording heads and a mechanism to manipulate the tapes. The UNISERVOs are controlled by the Central Computer. A read instruction directs the proper UNISERVO to connect its magnetic head to the read circuits and to move the tape past the magnetic head. Each pulse recorded on the magnetic tape generates an electrical pulse in the read circuits as the tape sweeps by the magnetic head. The information which was recorded on tape as a pattern of pulses appears in the read circuits as a train of electrical pulses corresponding to the pattern of pulses on the tape. The train of electrical pulses is delivered to an auxiliary memory in the input circuits without delaying the Central Computer. When desired, the data is transferred from the input storage to the high-speed memory. Information can be read from tape in either the forward or backward direction.

A write instruction directs the proper UNISERVO to connect its write circuit to the recording head and to move the tape past the recording head. The tape is automatically erased before any information is recorded on it. A train of electrical pulses representing the information to be written is delivered to an auxiliary memory in the output circuits, and as the Central Computer continues computation, the information is delivered to the write circuits of the UNISERVO from the output circuits of the Central Computer. A magnetic pulse is impressed on the tape whenever an electrical pulse appears in the write circuits. Information is written on tape in only the forward direction.

The UNIPRINTER translates information recorded on tape into a typewritten copy. The UNIPRINTER contains a standard electric typewriter. The typewriter keys are actuated in accordance with the pulse patterns on the tape. All keys, including upper and lower cases of the alphabet, punctuation marks, spaces, tabs, and carriage returns operate automatically. However, margin and tab stops are set by hand. A tape "edited" by the Central Computer for printing is complete in all details. Both the UNITYPER and the UNIPRINTER operate independently of the Central Computer.

Since much information is already recorded in punch-card files, a Card-to-Tape Converter has been designed. This device reads the holes photo-electrically and converts the information into pulse patterns on magnetic tape.

The tape is metallic and will not corrode. It is plated with a magnetic material. Information recorded on it may be stored permanently. A tape may be erased and reused when the information stored on it is of no further value. The tape is a few thousandths of an inch thick, one half inch wide, and of high tensile strength. Unityped tapes and tapes prepared by UNIVAC for printing on the UNIPRINTER are transcribed at a density of twenty characters to the inch. Tapes prepared by the Card-to-Tape Converter and tapes prepared by the Central Computer to be reused in computer operations are transcribed at a density of one hundred characters to the inch. One reel of tape, eight inches in diameter, contains approximately 1,500,000 characters at a density of one hundred to the inch.

The CENTRAL COMPUTER, the core of the UNIVAC System, performs the logical and arithmetic operations necessary to the solution of a problem. The Central Computer contains input and output circuits, a memory, arithmetic and logical circuits, and circuits to control the sequence of operations. The tape drives of the UNISERVOs are not synchronized with the internal operating circuits of the Central Computer. Information read from tape is received in the input circuits and auxiliary memory in readiness for the operations which are to be performed. Results are stored in the memory as they are

produced and are delivered to the auxiliary memory in the output circuits when they are to be recorded on tape.

The arithmetic and logical circuits perform basic operations at high speed. Essentially, these operations may be reduced to addition, subtraction, multiplication, division, comparison, and the selection and assemblage of data.

The control circuits link the instructions and the arithmetic units. They automatically sequence the operations of the computer as directed by the instructions.

A significant characteristic of UNIVAC is that it can read, write and compute simultaneously.

The UNIVAC is self-checking and the Central Computer contains two types of error detecting devices, companion checkers and odd-even checkers. The arithmetic circuits and most of the control circuits are duplicated, and the information in duplicate units is continuously compared. If a discrepancy occurs, the error circuits stop the computer and light a neon to indicate the unit in which the error occurred. Information is coded for the UNIVAC in such a way that the pulse pattern for each character must always contain an odd number of pulses. There are numerous odd-even checkers throughout the Central Computer. If a pulse pattern is detected which does not contain an odd number of pulses, the error circuits stop the computer and light a neon to indicate where the error was detected.

The Supervisory Control is an auxiliary device associated with the Central Computer. It contains the switches and signal lights for operating and servicing the computer. The error neons are located on the Supervisory Control panel. A printing unit and a keyboard similar to that of the UNITYPER are provided in the Supervisory Control. The operator maintains direct communication with the UNIVAC through the Supervisory Control. The operator can examine the contents of any part of the memory and every arithmetic register on an oscilloscope at the Supervisory Control. He can type in entries or corrections to the data in the computer, check intermediate results on the Supervisory Control Printer, and he can search for the origin of detected errors. The UNIVAC is designed to

operate automatically. The Supervisory Control provides the operator with an intimate contact with the computer so that he can follow in detail the operations performed in the computer and can interfere with the sequence of events if he deems this desirable.

SEC. 3. SOME APPLICATIONS OF THE UNIVAC SYSTEM

The UNIVAC System has proven its capabilities in the course of more than three UNIVAC-years of normal operation. It has solved numerous problems in each of four basic categories. These categories classify problem-solving according to the quantity of data processed and the amount of processing required for each unit of data.

Category I includes problems processing little input-output data and requiring little computation. Such problems, rarely repeated, demand more attention from the programmer than from the Central Computer and UNISERVOs. Exemplifying this type of problem is the calculation of the radiation pattern from a shaped antenna. This involved the evaluation of a definite integral by which the relative power radiated at each of a group of angles was computed from a system consisting of a feed horn working into a reflector "disk".

Problems involving little data but a large amount of processing fall into Category II. Here the heavy load is placed on the Central Computer, and little on the UNISERVOs. A Fourier summation was performed to produce tables for use in connection with an examination of the crystal structure of Banfield's and Kenyon's free radicals.

The third category puts heavy pressure on the operation of the UNISERVOs, but requires little effort on the part of the Central Computer. The selection of policies from a master file of an insurance company, for various types of processing, was so programmed that about 70,000 items could be examined each hour. From the master file arranged in order of district-policy number, those requiring premium notices were entered on one tape, those requiring either dividend or commission-processing on another, while a third tape received entries requiring special notifications. Control totals were maintained for checking purposes.

Category IV includes those problems which, requiring large quantities of input and output data, also require a large amount of processing. In general, such problems would not be attempted without the aid of a large computer.

Outstanding in this class are the operations of matrix algebra. Programs have been prepared and applied for the multiplication, for the inversion, and for editing the results of these operations, for matrices of orders up to 300 by 300. The matrices are partitioned into submatrices of order ten by ten or less. One set of programs (low-level) performs the multiplication and inversion of the submatrices; another set (high-level) treats the submatrices as elements of the large matrix by directing the application of the low-level routines. The elements are treated in floating-decimal form. The elimination method with successive iterations to improve the error matrix gives an inversion time of 50 hours for a 200 by 200 matrix (See references XVIII and XIX in the bibliography).

The examples cited represent problems from engineering design, scientific research, commerce, and mathematics, and testify to the flexibility and universality of the UNIVAC System.

SEC. 4. RESPONSIBILITIES OF THE PROGRAMMER

The types of problems that a programmer will be called upon to solve using a high-speed computer are multitudinous. Nevertheless, the task of the programmer can be conveniently broken down into four major parts--varying slightly from problem to problem. The first part includes the problem-analysis. The second part involves constructing a detailed logical outline of the solution of the problem and its translation into explicit instructions to the computer. The third part involves submitting the solution to the computer and obtaining the results. The fourth part is administrative including preparation of the report. These phases may occur in any chronological order, and one or more of these parts may be involved simultaneously in the handling of a problem; e.g., problem-analysis may very well continue from the start through the time that final results are obtained by the computer.

A more complete discussion of these four phases, which are the concern of a programmer, is included below as part of the introduction although much of the detail may be lost to a beginner. It is recommended, therefore, that at the conclusion of the first reading of the manual this section be studied again.

Problem-Analysis

Problem-analysis, first and foremost, requires that the problem be clearly defined. The programmer must be adamant in requiring a complete description of the problem, of the input data, of the form and volume of the desired output data, as well as of the computational or processing steps to be performed. In many cases a model, illustrating the problem, with sample input and output can be helpful. The problem analysis will include the study necessary to determine whether the current method of solution is to be accepted intact, or a new procedure better adapted to the computer is to be developed. Thus, the programmer determines what operations must be performed, and in what order they must be carried out.

Flow Charts and Coding

Proper analysis will enable the programmer to construct flow charts in sufficient detail to indicate the essential operations, logical as well as arithmetic, to be performed by the computer to achieve the solution to the problem. The flow-charts must contain enough detail to indicate clearly logical omissions, unnecessary operations, or errors in the contemplated "solution". The programmer must carefully review his flow-charts, for, if the flow-charts are correct, the problem might well be considered "solved".

However, this solution of the problem. in the form of flow-charts, is still meaningless to the computer. The operations indicated in the flow-charts must be translated into explicit instructions in the computer code.

After the coding is completed, it must be checked preferably by someone other than the original coder. The checking process cannot be overemphasized. It is imperative that the checker be thorough; not only must the coding itself be checked for errors, but the flow-charts must be examined for logical errors. Careful checking will save hours of grief in trying to run the problem on the computer. If the problem is mathematical in nature, a wise checker will run a numerical example on a desk calculator, following step-by-step each instruction in the coding. After the coding has been completed, the programmer must take a time-estimate for the problem - how long will it take to run on the computer?

Running the Problem on the Computer

After the coding has been completed and checked, it remains necessary to translate the written instructions from paper to the magnetic input tape of the computer. The instructions are transferred on the UNITYPER to tape and transcribed by a UNIPRINTER for proof-reading. It is essential that the programmer be satisfied beyond doubt that the correct information has been entered on the tape. If a problem involves a large amount of input data, it is wise to make arrangements to start the preparation of the input data well in advance of the expected date on which the problem is to be run. All reels of tape must be properly labeled, identifying the contents of the tapes. Space must be obtained for the filing of future tapes.

Well in advance, the programmer should determine the number of tapes required for each problem-run.

Before the problem is run on the computer, the programmer must compile a set of operating instructions. The responsibility of operating the computer is great. The operator runs problems not for a single programmer, but for many programmers and he cannot know intuitively the characteristics of each problem. The operating instructions should indicate explicitly and clearly all the information that the operator requires to run the problem. What tapes are mounted on which UNISERVOs and when? What special settings are initially required on the Supervisory Control? What "type-ins" are necessary and when? All of these questions and more must be answered in the operating instructions.

It is important for the programmer to realize from the beginning that it is the operator who actually runs the problem on the computer, performing all the physical operations at the Supervisory Control panel. The operator is a highly skilled and experienced man who is expert in the handling of the computer during a problem-run. The wise programmer will depend upon this knowledge and skill to run the problem efficiently with a minimum waste of valuable computer-time.

The programmer's chief role during a problem-run, especially during the initial run, is that of an observer, who satisfies himself that the problem is progressing properly. In the event that a programming error should arise, he should have some idea of where to begin looking for the error, and unless the error is almost immediately ascertained, the programmer may waste valuable computer time. Throughout a problem-run, a log of the operation is kept ... a record of past errors aids in avoiding the same pitfalls in the future.

After the computer run has been completed, and the results have been obtained on tape, the output tape must be filed for future references, the results must be printed, a complete report must be written. It is desirable to have the report of the problem, include coding, flow-charts, operating instructions, running times, discussions of new coding techniques, suggestions for improvement of the routine, and ideas for research. When all of this has been accomplished, the programmer may feel reasonably certain that his mission has been accomplished.

Chapter 2

Representation of Information

SEC. 1. PRELIMINARY DISCUSSION

In order for UNIVAC to perform the operations necessary to the solution of a problem, the computer must receive proper information. This information consists of certain defined arrangements of electronic pulses representing numeric, alphabetic and other typewriter symbols.

When a key of the UNITYPER is depressed, seven electric channels are activated and the resulting impulses create magnetic spots across the tape in accordance with the particular character involved.

It is the purpose of this chapter to present the relationships between the pulse patterns which UNIVAC accepts and the typewriter characters in which information is represented. To represent the variety of characters needed, UNIVAC uses the binary system of notation which is a primary characteristic fundamental to most digital computers. For introductory purposes it is sufficient to state that the absence of an electrical pulse is represented by zero and the presence of a pulse is represented by one.

The relationship between typewriter characters and pulse combinations is shown in the table on the last page of this chapter. Reference to this table will show, for example, that

0 is represented 1 00 0011
5 is represented 0 00 1000
M is represented 1 10 0111

The first pulse position contains the check pulse and is not shown in the table. A discussion of this pulse position may be found in Section 12 of this Chapter. The next two pulse positions are known as the "zone indicators" and it will be noticed that both are zero for numeric characters.

SEC. 2. DECIMAL AND BINARY SYSTEMS OF NOTATION

For a better understanding of the defined pulse combinations for numeric quantities employed by the UNIVAC system, it is advisable to discuss, more fully, the binary system of notation and its relation to the decimal system of notation. In the discussion to follow, numeric characters will be represented with only four pulse positions; the check pulse and "zone indicators" will be omitted. It must be remembered, however, that in the UNIVAC system all characters require seven pulse positions for representation.

In the Decimal Notation, the quantity three hundred fifty-nine is written in the decimal system, 359; i.e.,

$$(3\times10^2) + (5\times10^1) + (9\times10^0) = 300 + 50 + 9 = 359 \quad (1)$$

where $10^2 = 100$, $10^1 = 10$, $10^0 = 1$

The left hand side of the equality (1) shows the basic composition of the decimal, or "base ten" system. Each digit of (1) counting from right to left, is multiplied by successively higher powers of ten.

The binary system of notation uses two symbols (or digits) "0" and "1" as compared with the ten digits (0 through 9) used in the decimal system. For example, the decimal quantity nine, expressed in the binary system, appears as 1001. This is equivalent to stating that

$$(1\times2^3) + (0\times2^2) + (0\times2^1) + (1\times2^0) = 8+0+0+1 = 9 \quad (2)$$

where

$$2^3 = 8,\ 2^2 = 4,\ 2^1 = 2,\ 2^0 = 1.$$

The left-hand side of the equality shows the basic composition of the binary or "base two" system. Each digit of (2), counting from right to left, is multiplied by successively higher powers of two. Hence, the decimal quantities and their binary equivalents are as shown in the following table.

Table of Decimal and Binary Equivalents

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

SEC. 3. THE BINARY EXCESS-THREE SYSTEM

UNIVAC uses a modified binary code, called the excess-three system. Each decimal digit is represented, in binary, by its original value plus three. Reasons, justifying the use of the "excess-three" system rather than the pure binary system, are given below in Section 11. In later discussions the symbols "0" and "1" used in the excess-three system will be referred to as binary zero and binary 1.

Table of Decimal and Excess-Three Equivalents

Decimal	Decimal Excess-Three	Binary Excess-Three
0	3	0011
1	4	0100
2	5	0101
3	6	0110
4	7	0111
5	8	1000
6	9	1001
7	10	1010
8	11	1011
9	12	1100

Thus, 724 = 1010 0101 0111

where 7 = 1010
2 = 0101
4 = 0111

It should be clear that the UNIVAC system considers each digit of a quantity in binary excess-three notation which is somewhat different from the usual binary notation. For example the number 25 in the usual binary notations equals

$$16+8+1 = 1(2^4)+1(2^3)+0(2^2)+0(2^1)+1(2^0) = 11001$$

But in digital binary (excess-three) notation

2 5

25 = 0101 1000

SEC. 4. PRACTICE EXERCISES ON EXCESS-THREE REPRESENTATION

Represent the following quantities in binary excess-three notation. (Ignore zone indicators and check pulse).

1. 23

2. 407

3. 5891

What would be the 6-pulse code representation of the following quantities. (Ignore the check pulse).

4. B 100

5. T2 325

6. A- 000

SEC. 5. EXCESS-THREE COMPLEMENTS

If two positive quantities when added together, produce a power of ten, one is said to be the ten's complement of the other. For example, the ten's complement of 724 is 276, since 724 + 276 = 1000; the ten's complement of 51 is 49, for 51 + 49 = 100. Subtraction is performed by adding the complement of the subtrahend to the minuend. Carry resulting from the sum of the most significant digits is ignored. For example,

892 - 724 = 168,

using complements, 892 + 276 = 168.

In the excess-three system, complements on nine are obtained by substituting zeros for ones and ones for zeros. Hence, complements on ten. would be obtained by increasing by one the least significant decimal digit of the nine's complement

thus

4 = 0111

5 = 1000 = nine's complement of four

6 = 1001 = ten's complement of four

also

724 = 1010 0101 0111

275 = 0101 1010 1000 = nine's complement of 724

276 = 0101 1010 1001 = ten's complement of 724

Decimal	Excess-Three	Excess-Three Nine's Complement
0	0011	1100
1	0100	1011
2	0101	1010
3	0110	1001
4	0111	1000
5	1000	0111
6	1001	0110
7	1010	0101
8	1011	0100
9	1100	0011

It will be understood that when complements are referred to in succeeding exercises and discussions, tens' complements are implied.

SEC. 6. PRACTICE EXERCISES

Represent the complements of the following in excess-three notation (ignore zone indicators and check pulse)

1. 42

2. 436

3. 510

4. 7777

SEC. 7. EXCESS-THREE ADDITIONS

When two decimal digits are added which do not produce a ten's carry, the corresponding binary addition of their excess-three representations will exceed the correct sum by an excess-three; i.e., the sum is too great by an excess-three correction:

Decimal	Excess-Three	Decimal	Excess-Three
1	0100	4	0111
3	0110	5	1000
4	1010 = 7	9	1111 = 12

Hence, if no carry occurs when two digits are added, an excess-three must be subtracted (complement of excess-three, 1101, added) from that digit to produce the correct sum. In performing excess-three corrections, no carry is executed from decimal digit to decimal digit.

When two decimal digits are added to produce a ten's carry, the corresponding binary addition of their excess-three representations will produce a carry from the fourth binary digit position. Thus, the excess-three correction is missing from the sum digit; i.e., the sum is deficient by an excess-three correction:

Decimal	Excess-Three	Decimal	Excess-Three
8	1011	9	1100
7	1010	9	1100
Sum 5	0101 = 2	8	1000 = 5
Carry			
1	1	1	1

Hence, if a carry occurs when two digits are added, an excess-three must be added to that digit to produce the correct sum.

Example 1. Add 592 and 257.

592 =	1000	1100	0101	
257 =	0101	1000	1010	
	1101	0100	1111	
	1			carry
	1110	0100	1111	
		0011		correction if carry, + XS3
	1101		1101	correction if no carry, -XS3
849 =	1011	0111	1100	

SEC. 8. PRACTICE EXERCISES

Perform the following additions by means of the binary excess-three method.

1. Add 3 and 4.

2. Add 9 and 5.

3. Add 25 and 40.

4. Add 18 and 46.

5. Add 478 and 903.

SEC. 9. EXCESS-THREE SUBTRACTION

Excess-three differences are obtained by adding the complement of the quantity, smaller in absolute magnitude, to the larger, and appending the sign of the larger.

Example 1: 72 - 34 = 38, using complements 72 + 66 = 38. In the excess-three system,

	72 =	1010	0101	
	34 =	0110	0111	
complement of	34 =	1001	1001.	
Hence,	72 =	1010	0101	
complement of	34 =	1001	1001	
		0011	1110	
	1			carry
		0011		correction if carry, + XS3
			1101	correction if no carry, -XS3
	38 =	0110	1011	

SEC. 10. PRACTICE EXERCISES

Perform the following subtractions by means of the excess-three method. Add the complement of the smaller quantity to the larger quantity and append the sign of the larger.

1. 8 - 2

2. 82 - 55

3. 100 - 17

4. 325 - 109

5. 109 - 325

SEC. 11. ADVANTAGES OF BINARY EXCESS-THREE SYSTEM

Two of the advantages in using the binary excess-three system are:

(a) It is electronically easy to represent complements on ten in this system.

(b) A "carry" in the decimal system will produce a "carry" in the binary excess-three system, (e.g.) the addition of 6 and 5 in the three systems are:

Decimal	Binary	Binary Excess-Three
6	0110	1001
5	0101	1000
1	1011	0001
carry 1	no carry	carry 1

SEC. 12. SEVEN-PULSE CODE

The seventh pulse position which is added will contain a "check-pulse" designed to detect the gain or loss of a binary pulse. The check pulse is present, or absent, according as it is, or is not, necessary to make the number of pulses representing any character odd.

For example, A = 01 0100 requires a check pulse in order that an odd number of pulses be present, A = 1 01 0100; so also 6 = 00 1001 becomes 6 = 1 00 1001. On the other hand, no check pulse is required by C = 01 0110 which becomes C = 0 01 0110. Frequent checks are made throughout the computer circuits to insure that each character is represented by an odd number of pulses.

SEC. 13. INTERNAL MEMORY

The internal memory of the UNIVAC consists of acoustic delay lines. It contains 100 channels, each storing ten words. The "memory locations" of the 1000 words are numbered from 000 to 999. Every five seconds the entire content of the memory is automatically checked to insure the continued correctness of the stored information.

Transfer of data into a memory location automatically erases any information previously stored in that location. However, reading from a memory location does not destroy its contents. The symbol, ( ), is used to mean "the contents of"; i.e., (m) = the contents of memory location m in the computer, m being any number from 000 to 999.

SEC. 14. A COMPUTER "WORD"

Each memory location holds one "word" of information consisting of twelve characters. A "word" of information can be coded to take one of two forms:

(a) It may consist of twelve characters, representing a numeric quantity or other data to be processed. When it is a numeric quantity, the twelve characters in the word are the algebraic sign, followed by eleven decimal digits. A "zero" in the sign position represents a plus sign. The computer in performing multiplication and division, considers the decimal point to lie immediately to the right of the sign position. Thus, all quantities X, are treated as falling in the range $-1 < X < +1$. It will be seen later how quantities outside this range are handled.

(b) Or it may take the form of two "instructions" to the computer (e.g.)

B0$\underline{0}$120 C0$\underline{0}$185

Each instruction consists of six computer digits. The first two characters in each instruction designate the operation to be performed (and will be defined in succeeding chapters) and the fourth, fifth and sixth digits in each instruction designate a memory location. The third digit (underlined and usually not written) is not decoded.

The computer performs the two instructions serially; the right instruction is executed after the left instruction has been accomplished.

SEC. 15. DIGITAL POSITIONS

In later discussions, reference will be made to digital positions and, hence, some comment on this terminology is in order. Consider the twelve digital positions of a computer word,

$\underline{1}$ $\underline{2}$ $\underline{3}$ $\underline{4}$ $\underline{5}$ $\underline{6}$ $\underline{7}$ $\underline{8}$ $\underline{9}$ $\underline{10}$ 11 12

The location labeled $\underline{1}$ is the first digit position and contains the sign of a numeric quantity. The location labeled $\underline{2}$ is referred to as the most significant digit (MSD) position and location $\underline{12}$, the least significant digit (LSD) position. When discussing non-numeric words, it is usually better to refer to the digital positions 1 to 12.

SEC. 16. PRACTICE EXERCISES

Using the seven-pulse code, in exercises 1 to 4, represent the quantities:

1. A 425

2. R 310 U 100

3. $\not{x}$

4. A decimal zero.

Represent the following quantities in the seven-pulse code, perform the operations indicated and represent the result in seven pulses.

5. 5076 + 2438

6. 3247 - 1066.

SECTION 17 - UNIVAC PULSE CODE

ZONE	XS3															
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	ɣ	Δ	-	0	1	2	3	4	5	6	7	8	9			
01	ʀ	,	.	;	A	B	C	D	E	F	G	H	I			
10	ɪ		☒	/	J	K	L	M	N	O	P	Q	R	ʟ		ʊ
11	ᵽ	ᵦ			+	S	T	U	V	W	X	Y	Z	ø		

ɣ = IGNORE
Δ = SPACE
ʀ = CARRIAGE RETURN
ɪ = TAB
ʟ = SHIFT LOCK

ʊ = UNSHIFT
ø = ONE SHIFT (SINGLE SHIFT)
ᵽ = PRINTER STOP
ᵦ = PRINTER BREAKPOINT STOP
☒ = NOT AVAILABLE (USED INTERNALLY)
BLANK SQUARES NOT USED.

NOTE - IN GENERAL, UNITYPER HAS A STANDARD KEYBOARD, BUT OTHER TYPEWRITER SYMBOLS CAN BE, AND HAVE BEEN PROVIDED.

Chapter 3

Registers

SEC. 1. PRELIMINARY DISCUSSION

In Chapter 2 it was stated that the internal memory of UNIVAC consists of memory locations capable of storing 1000 words. UNIVAC also employs certain additional "storage facilities" called registers, the functions of which are quite different from those of memory locations.

When instructions and other data are transferred to the computer from magnetic tape, they are placed into memory locations, and retained there until called upon to take part in the procedures. As these data are required for computation and other operations, they are processed through the registers. These registers are used, then, to

(a) Transfer data between memory locations.
(b) Perform arithmetic and control operations.

It is the purpose of this chapter to discuss these registers and define the part they play in the processing of instructions and other data. In Chapter 9, the role of registers in the total logical pattern will be presented. It should be noted, first, that a register is said to be "erased" when it contains binary zeros. A register is said to be "cleared" when decimal zeros (in the excess-three code) replace its previous contents.

Several registers are required for temporary storage of data being processed by the computer. Two of these are one-word registers used for sequencing operations (CC,CR); four are one-word registers used for arithmetic and logical operations (rA, rX, rL, rF); four are multi-word registers used to transfer data (rV, rY, rI, r0).

It has been stated that transfer of data into a memory location automatically erases any information previously stored in that location. Similarly, a transfer of data into a register (except rI) erases information previously stored. However, unlike memory locations, reading from a register may, depending on the instructions, also clear the register of its previous contents. The action in rI, under such conditions, will be discussed in Chapter 8.

Throughout this manual, the symbol "m" represents a memory location number.

SEC. 2. REGISTERS CC AND CR

These two registers are concerned with the sequencing of control operations.

The control counter (CC) stores the number of the memory location containing the next pair of instructions to be executed.

The control register (CR) stores the current pair of instructions.

Further discussion of these two registers will be delayed until Chapter 8.

SEC. 3. ONE-WORD REGISTERS

The four registers, rA, rX, rL, and rF, are duplicated within the computer. The contents of the duplicated registers are continuously compared by checking circuits which immediately detect any discrepancy between the duplicated quantities.

Register A is used for:

(a) one-word transfers
(b) storing the addend (minuend) in addition (subtraction)
(c) retaining a partial or complete algebraic sum
(d) storing the more significant half of a twenty-two digit product or a rounded eleven digit product after multiplication
(e) storing the dividend at the start of division
(f) storing the rounded quotient after division
(g) retaining a quantity to be shifted right or left, and performing the shift
(h) assembling extracted quantities

(i) storing one component of a comparison

Register X is used for:

(a) one-word transfers
(b) storing the augend (subtrahend) in addition (subtraction)
(c) storing the multiplier during multiplication
(d) storing the less significant half of a twenty-two digit product after multiplication
(e) storing the unrounded quotient after division

Register L is used for:

(a) storing the multiplicand during multiplication
(b) storing the divisor during division
(c) storing one component of a comparison

Register F is used for:

(a) one-word transfers
(b) storing the extractor
(c) storing three times the absolute magnitude of the multiplicand during multiplication.

SEC. 4. MULTI-WORD REGISTERS

Register V is used for two-word transfers

Register Y is used for ten-word transfers

Register I is used to assemble "one block", 60 words, read from tape for transfer into the memory.

Register O is used to store one block from the memory until written on tape.

SEC. 5. ONE-WORD TRANSFERS

Register A

Instructions	
Bm	Erase rA and rX; transfer (m) to rA and rX.
Cm	Transfer (rA) to m; clear rA to decimal zeros
Hm	Transfer (rA) to m; do not alter rA
Km	Transfer (rA) to rL clear rA; ignore m.

A complete one-word transfer using rA involves two instructions; (a) transferring a quantity from the memory to rA, and (b) transferring the quantity from rA to the memory.

Example 1: (050) = a. Transfer "a" to 051 and 052.

Mem. Loc.	Instruction		Remarks
020	B 050		a --> rA and rX
		H 051	(rA) = a --> 051; (rA) = a
021	C 052		(rA) = a --> 052; 0 --> rA

Register X

Instructions	
Bm	Erase rA and rX; transfer (m) to rA and rX.
Lm	Erase rL and rX; transfer (m) to rL and rX.
Jm	Transfer (rX) to m; do not erase rX.

A complete one-word transfer using rX involves two instructions; (a) transferring a quantity from the memory to rX, and (b) transferring the quantity from rX to the memory.

Example 1: (050) = a. Transfer "a" to 051 and 052.

Mem. Loc.	Instruction	Remarks
020	B 050	a --> rA and rX
	C 051	(rA) = a --> 051; 0 --> rA
021	J 052	(rX) = a --> 052; (rX) = a.

Example 2: (050) = a. Transfer "a" to 051 without disturbing (rA)

Mem. Loc.	Instruction	Remarks
020	L 050	a --> rL and rX
	J 051	(rX) = a --> 051; (rX) = a.

Register F

Instructions	
Fm	Erase rF; transfer (m) to rF.
Gm	Transfer (rF) to m; do not erase rF.

A complete one-word transfer using rF involves two instructions; (a) transferring a quantity from the memory to rF, and (b) transferring the quantity from rF to the memory.

Example 1: (050) = a. Transfer "a" to 051, without disturbing (rA) or(rL).

Mem. Loc.	Instruction		Remarks
020	F 050		a --> rF
		G 051	(rF) = a --> 051; (rF) = a.

Interchange: Two quantities may be interchanged by means of one-word transfers.

Example 1: (050) = a. (051) = b. Interchange "a" and "b".

Mem. Loc.	Instruction		Remarks
020	B 050		a --> rA and rX
		L 051	b --> rL and rX
021	C 051		(rA) = a --> 051; 0 --> rA
		J 050	(rX) = b --> 050; (rX) = b.

SEC. 6. MULTI-WORD TRANSFERS

All multi-word transfers erase the registers or memory locations to which they are directed with the exception of rI.

Register V

Instructions

Vm Erase rV; transfer two consecutive words, starting with m, to rV; m is usually a multiple of two. For other cases see paragraphs to follow.

Wm Transfer (rV) to two consecutive memory locations, starting with m; do not erase rV; m is usually a multiple of two. For other cases see paragraphs to follow.

A complete two-word transfer using rV involves the two instructions: (a) transferring two successive words from the memory to rV, and (b) transferring the two quantities from rV to the memory.

Example 1: (050) = a, (051) = b. Transfer "a" and "b"to 096 and 097 respectively.

```
Mem.
Loc.          Instruction      Remarks

020           V 050            a, b --> rV
                      W  096   a --> 096; b --> 097; (rV) = a, b.
```

If the m in both the Vm and Wm instructions is odd and the least significant digit is not equal to nine, the instructions behave as in the following example.

Example 2: (051) = a, (052) = b. Transfer "a" and "b" to 063 and 064 respectively.

```
Mem.
Loc.          Instruction      Remarks

020           V  051           a, b --> rV
                      W  063   a--> 063; b --> 064;
                               (rV) = a, b.
```

If the m in one instruction is odd (least significant digit not equal to nine), and the m of the other instruction is even, the two words are transferred in reversed order.

Example 3: (051) = a, (052) = b. Transfer "b" followed by "a" to 054 and 055 respectively.

```
Mem.
Loc.          Instructions     Remarks

020           V  051           a, b --> rV
                      W  054   b --> 054; a --> 055;
                               (rV) = a, b.
```

Example 4: (050) = a, (051) = b. Transfer "b" followed by "a" to 063 and 064 respectively.

```
Mem.
Loc.        Instruction        Remarks

020         V  050             a, b --> rV
                W  063         b --> 063; a --> 064; (rV)=a, b
```

If the m in a Vm or Wm instruction has a nine as its least significant digit, the instruction will transfer from, or to, the last and first words in the ten-word memory channel.

Example 5: (050) = a, (059) = b. Transfer "a" followed by "b" to 100 and 101.

```
Mem.
Loc.        Instruction        Remarks

020         V  059             b, a --> rV
                W  100         a --> 100; b --> 101; (rV) = b,a.
```

Register Y

Instructions

Ym	Erase rY; transfer ten consecutive words starting with m, to rY; m should be an integral multiple of ten
Zm	Transfer (rY) to ten consecutive memory locations starting with m; do not erase rY; m should be an integral multiple of ten

A complete ten-word transfer using rY involves two instructions; (a) transferring ten successive words from the memory to rY, and (b) transferring the ten quantities from rY to the memory.

Example 1: (050) = a_0, (051) = a_1,..., (059) = a_9. Transfer "a_0,...,a_9" to 100,...,109 respectively

Mem. Loc.	Instruction		Remarks
020	Y 050		a_0,..., a_9 --> rY
	Z 100		a_0 --> 100,...,a_9 --> 109; (rY) = a_0,...,a_9.

When executing a Ym or Zm instruction, the least significant digit of m is ignored by the computer. The transfers operate on the integral multiples of ten. Thus, Y999 is equivalent to Y990, and Z784 to Z780.

Register I

Register I does not erase upon read in, but only upon transferral of its contents to the memory. The tape instructions reading into rI will be discussed in Chapter 8.

Instructions

30m	Transfer sixty words stored in rI to sixty consecutive memory locations, starting with
40m	m; m should be an integral multiple of ten; erase rI.

Register 0

Register 0 holds sixty words during a write instruction. Register 0 cannot be used independently of the tape instructions which will be discussed in Chapter 8.

SEC. 7. PRACTICE EXERCISES

Problems on the transfer of data

Given

(052) = x_1

=

(053) = x_2

(054) = x_3

(055) = x_4

(056) = x_5

(057) = x_6

(058) = x_7

(059) = x_8

(060) = x_9

(061) = x_{10}

In each of the following problems write the instructions to perform the operations indicated; start with memory location 020. Choose any unused memory locations for working storage.

1. Transfer quantity x_1 to 050 and 051 using

   (a) rA
   (b) rX
   (c) rF

2. Transfer quantity x_1 to 050 and 053 and place x_2 in 052.

3. Transfer x_1 to 053, x_2 to 054, x_3 to 053 and clear rA.

4. Transfer x_1 to 048 and 046, x_2 to 049 and 045 using the V, W instructions.

5. Move the given 10 quantities up two memory locations i.e., x_1 --> 050, x_2 --> 051, x_3 --> 052, ... x_{10} --> 059

Chapter 4

Fundamental Arithmetic Operations

SEC. I. PRELIMINARY DISCUSSION

The purpose of this chapter is to explain UNIVAC instructions for addition, subtraction, multiplication, division and to consider certain special problems closely allied to these operations. Before delving into these details, it is advisable to review some concepts, previously discussed.

(a) The internal memory of UNIVAC consists of 100 channels, each storing ten words. The "memory locations" of the 1000 words are numbered from 000 through 999.

(b) A "word" may have the form of two instructions or may be information composed of twelve typewriter characters.

(c) The instructions in a "computer word" are executed serially, first the left instruction followed by the right instruction.

(d) The symbol (m) represents "the contents of memory location m."

(e) A transfer into a memory location erases any information previously stored in that location but reading from a memory location does not destroy its contents.

(f) There are four one-word registers - rA, rX, rL, rF; the multi-word registers are rV, rY, rI, r0.

(g) A complete transfer involves two instructions -- transferring a quantity from a memory location to a register, and then, transferring the quantity from the register to a memory location.

In the performance of arithmetic processes, certain special conditions may arise and must be recognized. Two of these problems will be mentioned, briefly, and considered in more detail in later discussions.

(a) The algebraic addition, subtraction or division of numeric quantities, considered as decimals by UNIVAC, may lead to results greater than plus one or less than minus one. This situation, called "overflow" must be handled by special techniques which will be described in Chapter 7.

(b) In the study of instructions for arithmetic manipulation, the question may arise as to computer responses to quantities that contain characters, other than numeric. A discussion of this problem will be found at the end of this chapter.

Finally, it will be noted that the instructions, previously defined, are restated in this chapter. A complete table of instructions may be found at the back of this manual and it is so placed for easy reference.

SEC. 2. ADDITION

Instructions

Am	Transfer (m) to rX; add (rX) to rA); deliver the sum to rA; do not erase rX.
Bm	Erase rA and rX; transfer (m) to rA and rX.
Cm	Transfer(rA) to m; clear rA.
Hm	Transfer (rA) to m; do not clear rA.
Jm	Transfer (rX) to m; do not erase rX.
Km	Transfer (rA) to rL; clear rA; ignore m.
Sm	Transfer -(m) to rX; add (rX) to (rA); deliver the difference to rA; do not erase rX.
Xm	Add (rX) to (rA); deliver the sum to rA; do not erase rX; ignore m.

A complete addition involves three instructions; (a) transferring the addend to rA, (b) transferring the augend to rX, adding (rX) to (rA), and delivering the sum to rA, and (c) transferring the sum from rA to the memory.

Example 1: (049) = x, (050) = y. Deliver the sum x+y = z to 051.

```
Mem.
Loc.     Instruction      Remarks

020  B   049              x --> rA and rX
             A 050        y --> rX; x+y = z --> rA
021  C   051              (rA) = z --> 051; 0 --> rA.
```

Example 2: (049) = x, (050) = y. Deliver the sum p = x+y+y to 051.

```
Mem.
Loc.     Instruction      Remarks

020  B   049              x --> rA and rX
             A  050       y --> rX; x+y = z --> rA; (rX)=y
021  X   000              z+y = p --> rA
             C  051       (rA) = p --> 051; 0 --> rA.
```

Example 3: (049) = x, (050) = y, (051) = z. Deliver the sum x+y = p to 052, and the sum x+y+z = q to 053.

```
Mem.
Loc.     Instruction      Remarks

020  B   049              x --> rA and rX
             A  050       y --> rX; x+y = p --> rA; (rX)=y
021  H   052              (rA) = p --> 052; (rA) = p
             A  051       z --> rX; p+z = q --> rA; (rX)=z
022  C   053              (rA) = q --> 053; 0 --> rA.
```

SEC. 3. PRACTICE EXERCISES ON ADDITION

In the problems to follow:

(049) = x, (050) = y, (051) = z

use any memory location for working storage. Write the instructions to:

1. Deliver 3x to 060.
2. Deliver the sum x+2y to 060.
3. Deliver the sum 2x+3y to 060.
4. Deliver the sum 2x+y+3z to 060.
5. Deliver

   2x+y to 060

   3x+y to 061

   2x+2y to 062
6. Deliver

   x+y to 060

   y+z to 061

   z+x to 062

   2(x+y+z) to 062
7. Change (049) to 2x, (050) to 2y, (051) to 2z and send x, y and z to memory locations 060, 061, 062 respectively.

SEC. 4. SUBTRACTION

A complete subtraction involves three instructions; (a) transferring the minuend to rA, (b) transferring minus the subtrahend to rX, adding (rX) to (rA), and delivering the difference to rA, and (c) transferring the difference from rA to the memory.

Example 1: (049) = x, (050) = y. Deliver the difference x-y = z to 051.

Mem. Loc.	Instruction		Remarks
020	B 049		x --> rA and rX
	S 050		-y --> rX; x-y = z --> rA; (rX) =-y
021	C 051		(rA) = z --> 051; 0 --> rA.

Example 2: (049) = x, (050) = y, (051) = z. Deliver the difference p = x-y to 052 and the sum q = x-y+z to rL.

Mem. Loc.	Instruction		Remarks
020	B 049		x --> rA and rX
	S 050		-y --> rX; x-y = p --> rA; (rX)=-y
021	H 052		(rA) = p --> 052; (rA) = p
	A 051		z --> rX; p+z = q --> rA; (rX)=z
022	K 000		(rA) = q --> rL; 0 --> rA.

Example 3: (049) = b. It is desired to obtain -b when (a) rA is not cleared or (b) b = 0. This should be used with caution because there is a possibility of overflow depending on (rA).

Mem. Loc.	Instruction		Remarks
020	S 049		-b --> rX; (rA) -b --> rA
	J 050		(rX) = -b --> 050; (rX) = -b.

SEC. 5. PRACTICE EXERCISES ON SUBTRACTION

In the problems to follow:

(049) = x, (050) = y, (051) = z

provide the instructions to:

1. Deliver
   - (a) x-y to 060
   - (b) y-x to 061
   - (c) y-z to 062
2. Deliver
   - (a) -y to 060
   - (b) 2x-y to 061
   - (c) 2x-y+z to 062
3. Deliver
   - (a) x-2y to 060
   - (b) x-2z to 061

and send y to 049, z to 050, and x to 051.

SEC. 6. MULTIPLICATION

Instructions	
Jm	Transfer (rX) to m; do not erase rX.
Km	Transfer (rA) to rL; clear rA; ignore m.
Lm	Transfer (m) to rL and rX.
Mm	Transfer (m) to rX; multiply (rL) by (rX) = (m); deliver the rounded eleven digit product to rA; (rL) unaltered.
Nm	Transfer -(m) to rX; multiply (rL) by (rX) = - (m); deliver the rounded eleven digit product to rA; (rL) unaltered.
Pm	Transfer (m) to rX; multiply (rL) by (rX) = (m); deliver the more significant half of the twenty-two digit unrounded product to rA, the less significant half to rX; (rL) unaltered.

The instructions M, N, and P leave three times the absolute magnitude of the multiplicand in rF.

The instructions M and N leave the less significant half of the twenty-two digit product ±50000 000000 in rX.

Register A and rX are erased before the product is delivered. Hence, products may not be automatically accumulated.

The quantity having the fewer non-zero digits should be selected as the multiplier since multiplication by zeros consumes less time than multiplication by a non-zero digit.

A complete multiplication requires three instructions; (a) transferring the multiplicand to rL, (b) transferring the multiplier to rX, initiating the multiplication operation, and delivering the product to rA and rX, and (c) transferring the product from rA (and rX) to the memory.

Example 1: (049) = x, (050) = y. Deliver the eleven digit rounded product xy = z to 051.

```
Mem.
Loc.     Instruction     Remarks

020  L   049             x --> rL, and rX
         M   050         y --> rX; xy = z --> rA; (rL)=x
021  C   051             (rA) = z --> 051; 0 --> rA.
```

Example 2: (rA) = x, (050) = y. Deliver the eleven digit rounded product -xy = z to 051, leave z in rA.

```
Mem.
Loc.     Instruction     Remarks

020  K   000             (rA) = x --> rL
         N   050         -y --> rX; -xy = z --> rA; (rL)=x
021  H   051             (rA) = z --> 051; (rA) = z.
```

Example 3: (049) = L (050) = M. Deliver the twenty-two digit unrounded product P = LM to 051 and 052.

```
Mem.
Loc.     Instruction     Remarks

020  L   050             M --> rL and rX
         P   049         L --> rX; LM = P --> rA and rX;
                                          (rL) = M
021  J   052             (rX) = less significant half of
                                      P = P_1s --> 052
         C   051         (rA) = more significant half of
                                      P = P_Ms --> 051.
```

Example 4: (049) = L, (050) = M, (051) = N. Deliver LM = P_1 to 052 and NM = P_2 to 053, eleven digit unrounded products are desired.

Mem. Loc.		Instruction		Remarks
020	L	050		M --> rL and rX
			P 049	L --> rX; LM = P_1 --> rA; (rL)=M
021	C	052		(rA) = P_1 --> 052; 0 --> rA
			P 051	N --> rX; NM = P_2 --> rA; (rL)=M
022	C	053		(rA) = P_2 --> 053; 0 --> rA.

SEC. 7. PRACTICE EXERCISES ON MULTIPLICATION

(040) = x

(041) = y

(042) = Z

Write the instructions to deliver eleven digit rounded products.

1. (a) xy to 050

   (b) xz to 051

   (c) yz to 052

   (d) xyz to 053

Write the instructions to deliver eleven digit rounded products.

2. (a) x^2 = 050

   (b) $-x_3$ = 051

   (c) x^4 = 052

   (d) $-x^5$ = 053

Write the instructions to deliver eleven digit rounded products.

3. (a) $(x-y)^2$ to 050 and 051

(b) (x-y) (x-z) to 052 and 053

(c) $x(y-z)^2$ to 054 and 055.

SEC. 8. DIVISION

Instructions	
Jm	Transfer (rX) to m; do not erase rX.
Km	Transfer (rA) to rL; clear rA; ignore m.
Lm	Transfer (m) to rL and rX.
Dm	Transfer (m) to rA; divide (rA) = (m) by (rL); deliver the eleven digit rounded quotient to rA, and the eleven digit unrounded quotient to rX; (rL) unaltered.

Since all quantities, x, in the UNIVAC, must fall in the range $|x| < 1$, the absolute value of the dividend must be less than the absolute value of the divisor so that a proper division shall take place. If $|Dd| \geq |Dr|$, or if an attempt is made to divide by zero, an overflow will result (see Chapter 7).

A complete division requires three instructions; (a) transferring the divisor to rL, (b) transferring the dividend to rA and initiating the division operation, and delivering the quotient to rA and rX, (c) transferring the quotient from rA (or rX) to the memory.

Example 1: (049) = A, (050) = B. Deliver the eleven digit rounded quotient C = A/B to 051.

```
Mem.
Loc.     Instruction     Remarks

020  L   050             B --> rL and rX
         D   049         A --> rA; A/B = C --> rA; (rL)=B
021  C   051             (rA) = C --> 051; 0 --> rA.
```

Example 2: (049) = A, (050) = B. Deliver the eleven digit unrounded quotient C = A/B to 051.

```
Mem.
Loc.     Instruction     Remarks

020  L   050             B --> rL and rX
         D   049         A --> rA; A/B = C --> rX; (rL)=B
021  J   051             (rX) = C --> 051.
```

SEC. 9. PRACTICE EXERCISES ON DIVISION

Assume that overflow will not occur in these exercises.

(050) = x

(051) = y

(052) = z

Write the routines to send 11 digit rounded quotients.

1. (a) $\frac{x}{y}$ to 060 and 061

   (b) $\frac{x}{y}$ to 062

2. (a) $\frac{xy}{z}$ to 060

   (b) $\frac{x}{yz}$ to 061

In problem 2, also send the unrounded quotients to 062 and 063.

SEC. 10. REVIEW PRACTICE EXERCISES ON THE ARITHMETIC OPERATIONS

(050) = x

(051) = y

(052) = z

(053) = 1

Assume that adjustments have been made so that overflow will not occur in these exercises. Write the instructions to send:

1. x^2+3y^2-2z to 060

2. $(x+y)^2$ to 061

3. $\frac{1}{x} + \frac{1}{y} + \frac{1}{z}$ to 062

4. $\frac{x^2}{y} - \frac{2z}{y} - \frac{y}{4}$ to 063

SEC. 11. SPECIAL CONSIDERATION IN ARITHMETIC COMPUTATION ADDITION AND SUBTRACTION

When two numeric quantities are to be added (or subtracted), UNIVAC compares the signs of the quantities involved, performs the addition (or subtraction) on the terms digit by digit and delivers the correct result (including the sign) to rA. This, of course, presumes that overflow has not occurred and that decimal points have been properly aligned. Overflow will be considered, in some detail, in Chapter 7, and decimal point alignment is handled in Chapter 6. Also, it should be mentioned that the internal logic with which a computor performs these arithmetic operations is discussed in Chapter 9.

It is important, at this time, to reflect on the responses of the computer to add or subtract orders on "words"

which are not wholly numeric or/and which contain characters, other than 0 (plus) or - (minus), in the sign position.

Consider, first, the digital positions excluding the sign. In the addition and subtraction processes, the circuits by-pass alphabetic and typewriter characters and, hence, these quantities take precedence over numeric digits. A minus, not in the sign position, is treated as a digit. This means, for example, when a letter is added to (or subtracted from) a numeric digit the letter prevails and is shown in the result; the numeric character is ignored.

If any two characters, other than numeric, appear in a given digit the UNIVAC stops and shows an error neon. To illustrate, in a digit (other than sign) position:

4+3 = 7

4+B = B

4+(-) = 3

R+B = Error Stop

Following are two tables showing UNIVAC responses to add and subtract orders on non-numeric quantities.

ADDITION

ADDEND	AUGEND			
	D ≠ 0	+ = 0	−	C
− D ≠ 0	D or+0	D	D − 1	C
+ = 0	D	+	−	C
−	D − 1	−	Δ	C
C	C	C	C	E

D = DIGIT
C = CHARACTER
Δ = SPACE

SUBTRACTION

SUBTRAHEND	MINUEND			
	D ≠ 0	+ = 0	−	C
D ≠ 0	D or+0	−D	−D−1	C
+ = 0	D	+	−1	C
−	D +1*	1	+	C
C	C	C	C	E

D = DIGIT
C = CHARACTER
Δ = SPACE
E = ERROR STOP

*If a minus sign is subtracted from a nine the result is a "ten". Since this pulse code (0 00 1101) does not represent a character on the UNIPRINTER, it must be properly summed by adding decimal zero. As illustrated by the following examples:

Example 1: Subtract 0---- from 09999.

9999 =	1100	1100	1100	1100	
complement ---- =	1101	1101	1101	1110	
	1010	1010	1010	1010	
	0011	0011	0011	0011	correction if
"10"10"10"10" =	1101	1101	1101	1101	carry, + XS3
0000 =	0011	0011	0011	0011	
	0001	0001	0001	0000	
	0011	0011	0011	0011	correction if
1110 =	0100	0100	0100	0011	carry, + XS3

Example 2: Subtract 0---- from 05394.

5394 =	1000	0110	1100	0111	
complement ---- =	1101	1101	1101	1110	
	0110	0100	1010	0101	
	0011	0011	0011	0011	correction if
64"10"5 =	1001	0111	1101	1000	carry, + XS3
00 0 0 =	0011	0011	0011	0011	
	1100	1011	0000	1011	
	1101	1101	0011	1101	correction
6'50'5 =	1001	1000	0011	1000	

In the sign position, the circuits by-pass all symbols except a plus (0) or minus sign, and, hence, the symbols take precedence over the signs. If any combination of symbols other than the two signs appears in the sign position, an error stop results. In the case of subtraction, a digit or character subtracted from a sign leads to a different digit or character in the result. This is occasioned by the precedure, inherent in UNIVAC design which adds 1 00 0001, without binary carry, to the character in the sign position of the subtrahend. To illustrate:

When subtracting a plus quantity	0 =	1 00 0011
the computer adds		1 00 0001
the result is minus		0 00 0010
When subtracting a negative quantity	- =	0 00 0010
the computer adds		1 00 0001
the result is plus	0 =	0 01 0101
When subtracting B	B =	0 01 0101
the computer adds		1 00 0001
the result is A	A =	1 01 0100

When subtracting A the computer adds The result is B

A = 1 01 0100
1 00 0001
B = 0 01 0101

Therefore, in the sign position

0 + 0 = 0

(-) + (-) = (-)

B + 0 = B

B + R = Error Stop

0 - B = A

Below is a table showing, in part, the result in the sign position of subtract and add orders. A more complete representation of the Sm instruction in the sign column is shown on page 238 in the Appendix.

ADDEND OR SUBTRAHEND	AUGEND OR MINUEND			
	D ≠ 0	+ = 0	-	C
D ≠ 0	E	*D	*D	E
+ = 0	D	+	±	C
-	D	±	-	C
C	E	*C	*C	E

D= DIGIT
C= CHARACTER
E= ERROR STOP

*The effect of subtraction is described in the previous paragraph.

A negative zero cannot be obtained as a result of an addition or subtraction except in the cases -0 + (-0) = -0 and -0 - (0) = -0. In all other cases, if two quantities with opposite signs are equal, the positive sign is appended to the difference.

Multiplication and Division

A digit or a character in the sign position of a factor in multiplication or division is treated as a plus sign.

Furthermore, alphabetic characters in the other digital positions of the multiplicand, multiplier, dividend or divisor do not stop the UNIVAC. Since, in the multiplication and division processes, the zone indicators are ignored, all characters are treated as numeric digits. The behavior of such symbols as ɨ and Δ may be determined by carrying out these operations using the excess-three code. The following pulse combinations may appear in such a procedure:

Excess-Three	Decimal Equivalent
0000	-3 or 13, ɨ = ignore
0001	-2 or 14, Δ = space
0010	-1 or 15, - = minus sign
1101	10
1110	11
1111	12

Table 2, in the Appendix, shows the results of multiplying digits and characters.

SEC. 12. PRACTICE EXERCISES

1. Subtract 0---- from 03279

2. Show that, in the sign position, an "S" instruction on

   2 will give a 1

   H will give a G

3. Add where possible to B99325 C00100

   (a) 000001 000010

   (b) ------ ------

   (c) OZZZZZ OZZZZZ

   (d) ZZZZZZ ZZZZZZ

4. From Table 2 in the appendix determine the result of multiplying the digits.

   (a) E by D (Not in sign position)

   (b) 3 by X (Not in sign position)

   (c) X by 3 (Not in sign position)

   (d) 0 by B In the sign position

Chapter 5

Arrangement of Information

SEC. I. POSITIONING DECIMAL POINTS

Preliminary Discussion

Before presenting the instructions planned for this chapter, it is advisable to discuss the problem of positioning the decimal points of numeric quantities. When two words are acted upon arithmetically, UNIVAC will process the quantities by performing addition, subtraction, multiplication and division in each digital position. It is essential that the decimal point in the result of each computation be predetermined and that the quantities be so placed as to produce the desired result after calculation.

First of all, it will be recalled that UNIVAC recognizes a decimal point after the sign position and, therefore, other decimal points are assumed by the programmer. For example, if the inspection of a memory location reveals the "word" 000045824000 in storage, representing the quantity 458.24, the programmer would prepare for arithmetic manipulation by considering the stored quantity to be 0.00045 8 24000. The decimal point, although not shown in the composition of the word in storage, is inherent in the computer design, and the caret is conceived to indicate the location of the actual decimal point.

Decimal Point in Addition and Subtraction

When adding and subtracting two quantities the programmer must be certain that the units, tens and hundreds digits etc. of one quantity are in the same digital positions as those of the other. The shift order, described in this chapter, may be necessary to effect the alignment of digital positions.

POSITIONING OF DECIMAL POINT IN MULTIPLICATION

In order to determine the location of the decimal point in the product z = xy, the computer quantities are multiplied and the digital positions of the digits in the product are obtained. e.g. If x = 40 and y = 0.003

and x in the computer is $0.00040_{\wedge}00000$

and y in the computer is $0.0_{\wedge}0030000000$

Then z in the computer will be $0.000000_{\wedge}12000$

The number of digital places between the "two decimals" in the product, z, equals the sum of the corresponding digital places in the multipliers, x and y.

Note further that if

x in the computer is $0.0000000040_{\wedge}0$

and y in the computer is $0.000_{\wedge}00300000$

Then by the "P" instruction, z, the product, would be delivered as follows,

(rA) = 0.00000000000, (rX) = $0.00_{\wedge}120000000$

Note that the sign position in rX is ignored in the decimal point location.

Positioning of Decimal Point in Division

Consider the problem of determining the location of the decimal in the quotient x = z/y. First, it must be understood that for proper division the |y| as stored in the computer must be greater than |z| as stored in the computer. If this condition does not hold, it will be seen later in this chapter, that the shift instruction can be used to rectify this. Assuming that the condition here stated is satisfied, the computer quantity y is divided into the computer

quantity z and the digital positions of the digits in the quotient are determined. e.g. If y = 0.003 and z = 0.12

and z in the computer is $0.000000_{\wedge}1200$ (Dividend)

and y in the computer is $0.0_{\wedge}003000000$ (Divisor)

then x in the computer will be $0.00040_{\wedge}00000$ (Quotient)

The number of digital places between the "two decimals" in the quotient, x, equals the difference between the corresponding digital places in the dividend, z, and the divisor, y.

Exercises demonstrating these techniques, especially as they relate to the new instructions presented in this chapter, will be encountered.

The instructions described in this chapter supplement those explained in Chapters 3 and 4, and, hence, it is again suggested that the UNIVAC Code Sheet, stored in the back of this booklet, be used to help identify instructions previously presented.

SEC. 2. USE OF THE "FLOATING DECIMAL POINT"

In the arithmetic manipulation of numeric quantities, it is convenient, often, to resort to a "floating decimal point" routine. In such procedures all numeric quantities may be represented as follows:

Digital Position	Contains
1	Algebraic sign of quantity
2 through 10	The numeric quantity with the first non-zero digit in the second digit position
11 and 12	Exponent of required power of ten plus 49. The use of 49 is made to facilitate the representation of negative exponents.

For example: $15,379 = +.15379 \times 10^5$

appears as 015379 000054

Also, $0.0015379 = .15379 \times 10^{-2}$

Appears as 015379000047

When two quantities, represented in floating decimal point notation, are to be combined by any one of the four fundamental operations of arithmetic, certain adjustments may be necessary. The first step in performing anyone of the four fundamental operations is to separate the quantities from their exponents, storing them in memory locations. For example, consider the quantities $A \times 10^A$ and $B \times 10^B$ which are stored as

Aa = ± xxxxxxxxxxee a = ee

and Bb = ± xxxxxxxxxxff b = ff

A and a, B and b are separated, by methods to be discussed later in this chapter, and placed in storage.

If it is desired to obtain

$$C \times 10^C = A \times 10^A \pm B \times 10^B$$

it should be clear that these operations can be performed only if a = b. If a and b are equal, A and B are added (or subtracted) algebraically and their result C is obtained. Of course, c = a = b and the result Cc = ± xxxxxxxxxxgg (c = gg) can be stored. If however, a and b are unequal, for example, a < b, A is shifted to the right, as described later in this chapter, until a = b and then the operation, as explained above, can be executed.

In the multiplication

$C \times 10^C = (A \times 10^A)(B \times 10^B)$ and the product C = A x B and the sum c = a + b are determined and the quantity $C \times 10^C$ can be formed, in storage, in the floating decimal notation.

It should be clear that c must be reduced by 49 in order to represent the exponent of the product, in the excess-forty-nine code.

In the division

$$c \times 10^{c} = A \times 10^{A} / B \times 10^{B}$$

C = A/B and c = a - b. An adjustment will be necessary to represent the exponent (c) of the quotient in the excess-forty-nine code. Also, it should be evident that UNIVAC demands that $|A| < |B|$ and if this is not so, a routine must be instituted to rectify this situation.

SEC. 3. PRACTICE EXERCISES

(050) = 000640000000 = 6.4 = x

(051) = 000400000000 = 4.0 = y

Eleven digit-rounded results are required in the following exercises. Indicate decimal points and deliver.

1) $x(x+y)$ to 060

2) $\frac{x^2}{y}$ to 061

3) $-\frac{x^2+y^2}{y}$ to 062

4) x^2+y^2-2xy to 063

The following quantities are given in the floating decimal point notation. What numbers transcribed as fixed decimals, do they represent?

5) -64530000045

6) 070056000056

7) 020056000140

SEC. 4. SHIFT INSTRUCTIONS

The shift instructions are used largely to

(a) multiply and divide by powers of ten
(b) position quantities and decimal points
(c) obtain absolute magnitudes
(d) delete undesired information

Instructions

.nm	Shift all digits of rA, including the digit in the sign position, n digits to the right (abbreviated SR^*_N); drop the n least significant digits; supply n decimal zeros in the sign and most significant digit positions; ignore m. (Note n is any number from 1 to 9).
;nm	Shift all digits of rA, including the digit in the sign positions, n digits to the left, (abbreviated SL^*_N); drop the sign and n-1 most significant digit positions; supply n decimal zeros in the least significant digit positions; ignore m.
-nm	Shift all digits of rA, except the sign n digits to the right (abbreviated Srn); drop the n least significant digits; supply n decimal zeros following the sign; ignore m.
Onm	Shift all digits of rA, except the sign, n digits to the left (abbreviated SL_N); drop the n most significant digits (following the sign); supply n decimal zeros in the least significant digit positions; ignore m
OOm	"Skip" instruction; perform no operation, continue to the next instruction; ignore m

The instructions .Om, ;Om and -Om may not be used. In these cases, the computer will continue to shift until, at the end of two seconds, a stall is indicated.

A complete shift operation requires at least three instructions; (a) transferring the quantity to be shifted to rA, (b) shifting the quantity n places right or left, including, or not including, the sign, (c) transferring the shifted quantity from rA to the memory.

Example 1: (050) = x. Obtain |x| and store it in 051.

```
Mem.
Loc.     Instruction      Remarks

020      B  050           x --> rA and rX
                ;1 000    SL*1 (x) dropping the sign
021      .1 000           SR*1 (x); 0 --> sign position
                C  051    (rA) = |x| --> 051
```

Example 2: A word W of an input item contains the following information.

Digit Position	Information	
1	0	
2, 3, 4	type of policy	
5, 6	year	
7, 8	month	issue date
9, 10	day	
11, 12	rate basis	

The word is stored in 628. The issue date is to be isolated and delivered to memory location 145. Let W = $Ot_1t_2t\ y1y2\ m_1m_2{}^{d}{}_1{}^{d}{}_2{}^{r}{}_1{}^{r}{}_2$.

This subroutine is a part of the program to evaluate lapsed life insurance policies.

Mem. Loc.	Instruction	Remarks
020	B 628	W --> rA and rX
	.2 000	(rA) = 000 $t_1t_2t_3y_1y_2m_1m_2d_1d_2$
021	;6 000	(rA) = $y_1y_2m_1m_2d_1d_2$ 000000
	.4 000	(rA) = 0000 $y_1y_2m_1m_2d_1d_2$ 00
022	C 145	(rA) = issue date --> 145

Note: This result may also be obtained by extraction. See example 1 in Section 6 of this Chapter.

Example 3: (Illustration of "rounding off" procedure)

(050) = 0 48843270530 = x = .4884327053

(051) = 08 0000000000 = y = 8

(052) = 000000000005 for rounding out

Deliver $\frac{x}{y}$ 11 digit rounded quotient to 060

Mem. Loc.	Instruction	Remarks
020	00 000	
	L 051	8 --> rL
021	D 050	0.61058092566 --> rA
	A 052	Rounding 0.61058092571 --> rA
022	-1 000	(SR) 0.06105809257 --> rA
	C 060	

Example 4: It is desired to obtain 5 % of $15379.73 = A, rounded to dollars and cents, and deliver it to memory location 052.

(050) = 015379 730000 = A

(051) = 005000 000000 = 5%

(052) = 5 % of A

Mem. Loc.	Instruction		Remarks
020	B 051		005 --∠ rA and rX
		-4 000	SR4 (rA) = 0000005 = (rA)
021	C 051		(rA) --∠ 051
		L 050	A = 015379 730000 -- > rL and rX
022	M 051		000000 0768 99 --> rA
		04 000	SL4 (rA) = 000768 990000
023	C 052		5 o/o of A --> 052

SEC. 5. PRACTICE EXERCISES USING SHIFT INSTRUCTION

For use in exercises 1 to 4

(050) = -00452832000 = 452.832 = x

(051) = 090000000000= 9.0 = y

(052) = 000000704440= 70.444 = z

Write the instructions necessary to

1. Obtain x and store in 060.

2. Obtain x+z and store in 061.

3. Obtain x-z and store in 062.

4. Obtain x+y+z and store in 063.

5. Write the instructions to evaluate

$$x + x^2 + x^3 \quad \text{If } x = 0.01$$

Assume necessary constants in memory and obtain 11 digit unrounded result. Place the most significant digit of the answer in the M.S.D. position.

6. It is desired to take 6 % of $3,735.32 rounded to dollars and cents and deliver the result to memory location 060

(050) = 03725 320000

(051) = 0600000000000

7. In example 2 of Section 4, isolate the type of policy and the rate basis and put in memory locations 600 and 601

8. Assume that the

(050) = 004455143625

represents in the first six digital positions the price of a radio set and its coded description in digital positions 7 through 12. Provide the instructions for isolating the price into memory location 060.

SEC. 6. EXTRACT INSTRUCTIONS

The extract instruction is used to select, assemble, and delete information.

Instructions

Fm	Transfer (m) to rF. A "one" (or any character or digit whose least significant binary digit is zero) "specifies" an extraction. A "zero" (or any character or digit whose least significant binary digit is one) does not specify an extraction.
Em	Erase the digit positions of rA specified by rF. Insert in these positions the corresponding digits of (m).

A complete extract operation requires three or four instructions, (a) transferring the "extractor" to rF, (b) transferring the quantity to be extracted upon to rA, (c) extracting the specified digits from m into rA, and (d) transferring the result to the memory.

Example 1: Extract the "issue date" described in Example 2, Section 4 of this chapter.

(050) = 000011 111100 = E_1

(145) = issue date

(628) = W = word containing issue date

Mem. Loc.	Instruction		Remarks
020	F 050		E_1 = 000011 111100 --> rF
		K 000	0 --> rA; (rA) = rL
021	E 628		E_1(W) = 0000 $y_1y_2m_1m_2d_1d_2$
			00 --> rA
		C 145	(rA) = issue date --> 145

Example 2: The quantity 15,379 is represented, with a floating decimal point, as 015379 000054.

It is required to separate the exponent and digits of the quantity in preparation for arithmetic operations.

(050) = 015379 000054 = A'

(051) = 000000 000011 = E_1

(052) = A = digits of quantity

(053) = a = exponent of quantity

(054) = 000000 00049

(055) = decimal zeros

Mem. Loc.	Instruction		Remarks
020	F 051		E_1 = 000000 000011 --> rF
		k 000	0 --> rA; (rA) --> rL
021	E 050		E_1(A') = 000000 000054 --> rA
		S 054	-49 --> rX; 000000 000005 = a --> rA
022	C 053		(rA) = a --> 053; 0 --> rA
		B 050	A' = 015379 000054 --> rA
023	E 055		E_1 (055) --> A' = 015379 000000
		C 052	(rA) = A --> 052

Example 3: The sign of the quantity A is to be changed from plus to minus.

(049) = A > 0

(050) = -00000 000000 = E_1

Mem. Loc.	Instruction		Remarks
020	F 050		E_1 = -00000 000000 --> rF
		B 049	A --> rA and rX
021	E 050		E_1(E_1) = -A in rA
		C 049	(rA) = -A --> 049

SEC. 7. PRACTICE EXERCISES ON EXTRACT ORDERS

1) Write the instructions to obtain the absolute value of a quantity (A) by means of the extract order.

2) (050) = 027543000052

(051) = 048270000054

The quantities are represented in floating decimal point notation. Write the instructions to deliver the sum of these quantities to 060.

3) It is desired to assemble certain information concerning life insurance policies into one computer "Word", to be placed as follows

Information	Digital Positions	Code
Policy Account	2	N
Branch	3	B
Year	4, 5	Y_1, Y_2
Plan	6, 7, 8	P_1, P_2, P_3
Age	9, 10	A_1, A_2
Not Used	1, 11, 12	

When assembled in a computer word, it will appear as $ONBY_1Y_2P_1P_2P_3A_1A_200$. Assume that this information is stored, as part of more complete data, in 3 different memory locations as follows

$$(050) = XXXXXBXXXXXP_1$$

$$(051) = P_2P_3XXXA_1A_2XXXXX$$

$$(052) = XXXXXXY_1Y_2NXXX$$

Provide the instructions needed to assemble this data and place in memory locations 060.

Example 4: A word W of an input item contains the following information

Digit Position	Information
1, 2, 3, 4	Policy Number
5, 6, 7	Type of Policy
8, 9. 10, 11, 12	Premium

The word is stored in 050. The three items in this word are to be isolated and sent to memory locations 060, 061, 062. Provide the necessary instructions.

(080) = 1111 00000000 E_1

(081) = 000011100000 E_2

(082) = 000000011111 E_3

CHAPTER 6

TRANSFER OF CONTROL

SEC. 1. PRELIMINARY DISCUSSION

In this chapter, instructions will be defined which will transfer control conditionally and unconditionally. Both types of control transfer are extremely useful in the development of computer programs. For example, routines which lead to iterative patterns are important in computer processes and are direct applications of these new instructions. It will be useful to the reader if the concept of iteration is developed more fully prior to the presentation of the actual instructions.

The foregoing chapters contain the procedures necessary for simple computer processing such as transfer of data, arithmetic operations, extracting and shifting. Routines, composed of these instructions often require many successive repetitions.

To take an elementary illustration, consider the problem of evaluating

$$y_i = ax_i + b$$

where a and b are constants, and x_i represents nine different, but related, values of x. Assume that these values of x are $x_i = (0.1, 0.2, 0.3 \ldots 0.9)$. It is apparent that the arithmetic processes required to obtain the nine corresponding values of y_i are repeated exactly nine times, using for x_i the nine values given. The instructions for transfer of control, presented herein, enable the programmer to encode for this iterative situation. The reader will observe additional applications of these instructions in the examples provided.

SEC. 2. THE STOP INSTRUCTION

The stop instruction is used to terminate a computation.

INSTRUCTION	
90m	Stop UNIVAC operation. Light "stop" neon.

SEC. 3. INSTRUCTION FOR TRANSFER OF CONTROL

The sequence of instructions to the UNIVAC may be interrupted and altered both conditionally and unconditionally. This provides a method for entering, re-entering and exiting from subroutines. A "transfer of control to m" means that the next instruction pair is to be obtained from memory location m, instead of from the memory location in the normal sequence.

INSTRUCTIONS

Qm	If (rA) = (rL), transfer control to m.
Rm	If Rm is entered on line c, record [00 000 U (c+1)] in m.
Tm	If (rA) > (rL), transfer control to m.
Um	Transfer control to m.
OOm	Skip. Continue to the next instruction; ignore m.

SEC. 4. Q, T, U AS RIGHT HAND INSTRUCTIONS

Qm, Tm and Um are normally entered as the right hand instruction of a pair. For exceptions see examples in Section 5 of this chapter.

A complete test operation using the Qm and Tm instructions requires three operations; (a) transferring a quantity to rA, (b) transferring a quantity to rL, and (c) comparing (rA) with (rL).

Example 1: (049) = x, (050) = y. Transfer the algebraically greater quantity to 100.

```
Mem.
Loc.    Instruction        Remarks

020    00 000
              B  049        x --> rA and rX
021    L  050               y --> rL and rX
              T  023        If x > y, go to 023
- - - - - - - - - - -

022    J  100               If x ≤ y, (rX) = y --> 100
              U  024        go to 024
- - - - - - - - - - -

023    C  100               If x > y, (rA) = x --> 100;
              00 000        Skip              0 --> rA
- - - - - - - - - - -

024                         Continue routine
```

Example 2: (049) = x, (050) = y. If x ≥ y, enter a "1" from 040 in 051 and return to main routine; if x < y continue main routine.

```
Mem.
Loc.    Instruction        Remarks

020    00 000
              K  000        (rA) --> rL; 0 --> rA
021    S  050               -y --> rX; 0 - y --> rA
              K  000        (rA) = -y --> rL; 0 --> rA
022    S  049               -x --> rX; 0 - x --> rA
              T  024        If -x > -y (x < y), go to 024
- - - - - - - - - - -

023    F  040               If -x ≤ -y (x ≥ y), 1 --> rF
              G  051        (rF) = 1 --> 051
- - - - - - - - - - -

024                         Continue main routine
```

Example 3: (050) = x. If x > 0, continue routine A line 023. If x = 0, go to routine B line 061. If x < 0, go to routine C line 083.

```
Mem.
Loc.    Instruction        Remarks

020     00 000
                B  050     x --> rA and rX
021     K  000             x --> rL; 0 --> rA
                T  083     If 0 > x, go to 083 routine C
- - - - - - - - - - -

022     00 000             If 0 ≤ x, test for equality
                Q  061     If 0 = x, go to 061 routine B
- - - - - - - - - - -

023                        If 0 ≠ x; i.e., 0 < x, continue
                           routine A.
```

Example 4: (512) = 1, (614) = 0, (633) = d,

If d = 1, go to routine A location 235;
If d = 2, go to routine B location 249;
If d = 3, go to routine C location 255;
If d = 4, go to routine D location 574;
If d ≠ 1, 2, 3, or 4, stop computation

This subroutine was required to classify life insurance policies according to dividend preference.

```
Mem.
Loc.    Instruction        Remarks

229     L  614             0 --> rL and rX
                B  633     d --> rA and rX
230     S  512             -1 --> rX; d-1 --> rA; (rX) = -1
                Q  235     If d-1 = 0, go to 235
- - - - - - - - - - -

231     X  000             (rA) + (rX) = d-2 --> rA
                Q  249     If d-2 = 0, go to 249
- - - - - - - - - - -
```

```
232   X  000              (rA) + (rX) = d-3 --> rA
           Q  255         If d-3 = 0, go to 255
- - - - - - - - - - -

233   X  000              (rA) + (rX) = d-4 --> rA
           Q  574         If d-4 = 0, go to 574
- - - - - - - - - - -

234   00 000              If d ≠ 1, 2, 3, or 4
           90 000         Stop computation.
```

SEC. 5. Q, T, U AS LEFT HAND INSTRUCTIONS

If a Qm, Tm, or Um instruction is placed in the left half of a word, it will cause a transfer of control subject to the usual conditions, but the memory location to which the transfer is made will be that contained in the right hand instruction. Both instructions will be executed.

Example 1: E_1 = 000000 000011 = (049), (050) = A, (051) = B, (052) = C, (053) = a, (054) = b. If a > b, extract from A and deliver to C. If a ≤ b, extract from B and deliver to C.

```
Mem.
Loc.   Instruction        Remarks

020   00 000
           F  049         E1 = 000000 000011 --> rF
021   L  054              b --> rL and rX
           B  053         a --> rA and rX
022   T  000              If a > b, go to 024 after executing
           K  024         a --> rL; 0 --> rA          next K
- - - - - - - - - - -

023   E  051              E1 (B) --> rA
           U  025         go to 025
- - - - - - - - - - -

024   E  050              E1 (A) --> rA
           00 000         Skip
- - - - - - - - - - -

025   C  052              (rA) --> 052; 0 --> rA
```

Example 2: (049) = x, (050) = y, (051) = z, (052) = 1. If x + 1 = y go to routine A line 061. If x + 1 ≠ y and x + 2 = z, go to routine B location 732.

```
Mem.
Loc.    Instruction        Remarks

020    00 000
              L  050        y --> rL and rX
021    B  049               x --> rA and rX
              A  052        1 --> rX; x+1 --> rA; (rX) = 1
022    Q  000               If x+1 = y, go to 061, after
                            executing next X
              X  061        x+2 --> rA; (rX) = 1
- - - - - - - - - - -

023    L  051               z --> rL and rX
              Q  732        If x +2 = z, go to 732
- - - - - - - - - - -

024                         If x+2 ≠ z, continue main routine
```

Example 3: (050) = a, (051) = b. If a ≥ b control is to be transferred to a subroutine starting on location 243.

```
Mem.
Loc.    Instruction        Remarks

020    B  050               a --> rA
              L  051        b --> rL
021    T  000               If a > b transfer control to 243
              Q  243        If a = b transfer control to 243
```

When Qm and Tm instructions, in either order, are in the same instruction pair transfer of control takes place if (rA) ≥ (rL), and transfer is effected to the memory location indicated by the right hand instruction.

SEC. 6. COMPARISONS IN THE SIGN POSITION

So that the Tm instruction shall transfer control when (rA) is algebraically greater than (rL), the signs of both quantities must be in the sign position. If any character or digit other than a sign occupies the sign position, a

twelve digit comparison is made, and the transfer of control takes place if (rA) has a pulse code of greater magnitude than (rL). Thus, S > H, D > C, A > 8, 0 > -, - > Δ, Δ > ⅋; refer to the UNIVAC pulse code given in Chapter 2.

SEC. 7. THE R-U INSTRUCTIONS

The Um instruction is frequently used to transfer to a subroutine. If the Rm instruction is used in conjunction with the Um instruction, a transfer of control may be preset to return to the main routine upon completion of the subroutine.

Example 1: Illustration of the use of the R-U Instructions

Consider the following set of instructions

Mem. Loc.	Instruction	Remarks
020	R 150	Sends 000000 U 021 to 150
	U 100	Control to 100
- - - - - - - - - -		
021		Main Routine
022		
- - - - - - - - - -		
100		Subroutine A
101		
.		
.		
.		
150	00 000	
	U 021	Control to 021

Comments: The effect of the R-U instruction in memory location 020 is to

(a) Automatically place 00 000 U 021 in memory location 150

(b) Send control to memory location 100 for the development of subroutine A

(c) Send control back to memory location 021 to continue main routine.

Example 2: Two quantities in a "floating decimal point" routine together with their exponents are transferred to "working storage" locations. Control is transferred to the "floating decimal" addition routine. After the addition is completed, control is returned to the main routine and the result withdrawn from working storage.

```
          (002) = A  where A' = A x 10^a
          (003) = a
          (004) = B  where B' = B x 10^b              working
          (005) = b                                   storage
          (006) = C  where C' = C x 10^c = A' + B'
          (007) = c
          (026)      floating decimal addition routine.
through   (083)
          (050) = A
          (051) = a
          (052) = B
          (053) = b
          (054) = C
          (055) = c
```

Mem. Loc.	Instruction			Remarks
120	V	050		A, a --> rV
		W	002	A --> 002, a --> 003
121	V	052		B, b --> rV
		W	004	B --> 004, b --> 005
*122	R	083		record [00 000 U 123] in 083
		U	026	obtain next set of instructions from 026

```
026
 .                          Floating decimal addition routine
 .
082

083  [00 000
               U  123]      Return to main routine
123   V  006                C, c --> rV
               W  054       C --> 054, c --> 055
124                         Continue main routine.
```

*Note particularly the use of the R-U instructions.

SEC. 8. PRACTICE EXERCISES ON CONTROL TRANSFER

1. (050) = ±x

   (051) = ±y

   Write the instructions which will compare the absolute values of these quantities and send the larger to 060 and the smaller to 061.

2. (050) = x

   (051) = y

   (052) = 1

   If x < y, add one to x and transfer control to Routine A, line 020, if x > y stop computation.

3. Write the instructions to evaluate.

   $y_i = .5x_i + .004$

   for 10 values of x_i

   $x_1 = 0.01$, $x_2 = 0.02$, $x_3 = 0.03$ -----$x_{10} = 0.10$

(050) = 001000 000000

(051) = 050000 000000

(052) = 000400 000000

(053) = 010000 000000 (For testing)

Put the 10 values of y in memory locations 060 to 069.

Stop computer after storing y_{10}.

4. Six 6-digit numbers a, b, c, d, e, f occupy the last six digital positions in memory locations 050, 051, 052, 053, 054, 055 respectively. The contents of each memory location is in the form

000000 XXXXXX

It is required to set up a routine which will place the numeric quantities in these words in memory locations 090, 091, 092 in the form

ab, cd, ef

That is, the quantity a would occupy the first 6 digital positions of 090 and the quantity b the last 6 digital positions etc.

However, this is to be done only when the given number is less than 000000 100000 (in other words when the 7th digit is zero). If this is not so, zeros are to replace the number.

5. Consider any thirteen cards dealt to a player in a bridge game. Refer to these cards as a_1, a_2, ... a_{13} and place some appropriate representation of these cards in memory locations 040 to 052. Develop a routine which will determine the number of cards in each suit, and place these numbers in 060, 061, 062, 063.

6. In problem 5, also determine the number of honor cards (Jacks, Queens, Kings, Aces) and place in memory locations 064, 065, 066, 067.

Sec. 9. Breakpoint Instructions

The "breakpoint" instructions are used largely to check a program the first time it is run. Certain switches and buttons on the Supervisory Control are associated with the breakpoint instructions.

(a) Breakpoint Switch. Operated in conjunction with the ,m instruction. Two positions, "normal" and "breakpoint".

(b) Conditional Transfer Breakpoint Selector Buttons. Operate in conjunction with the Qnm and Tnm instructions. Twelve buttons, "Reset", "0", ... "9" "All". One or more of the buttons "0", ..., "9" may be depressed simultaneously.

(c) Conditional Transfer Switch. Operates in conjunction with the Qnm and Tnm instructions. Three positions, "Normal", "Transfer", and "No Transfer".

Instructions

,m Breakpoint stop; ignore m. If the Breakpoint Switch is in the normal position, interpret ,m as a skip instruction. If the Breakpoint Switch is in the breakpoint position, interpret as a stop instruction.

Qnm or Tnm Conditional transfer breakpoint stop. If the Conditional Transfer Breakpoint Selector Button "Reset" is used, the Qnm and Tnm instructions operate in the normal manner. If the button corresponding to n is depressed, the UNIVAC will stop after (rA) and (rL) have been compared by Qnm or Tnm, but before the transfer of control takes place. If the "All" button has been set, the UNIVAC will stop on all Qnm and Tnm instructions, after (rA) and (rL) have been compared, but before the transfer of control takes place.

The transfer of control indicated by the Qnm or Tnm instruction will or will not take place according to the position of the Conditional Transfer Switch. This switch, three position and non-locking, must be manually operated if positions (b) or (c) are desired.

(a) Normal. The Qnm and Tnm instructions operate in the normal manner.

(b) Transfer. The Qnm and Tnm instructions operate to transfer control regardless of the relative magnitudes of the quantities in rA and rL.

(c) No-Transfer. The Qnm and Tnm instruction do not transfer control regardless of the magnitudes of the quantities in rA and rL.

The switch may be moved to the transfer or no-transfer position after a stop caused by a conditional transfer breakpoint in order to transfer or not transfer control. A neon (CT) indicates whether or not the magnitudes of rA and rL had indicated a control transfer.

Chapter 7

Overflow

Sec. 1. Preliminary Discussion

It will be recalled that numeric quantities are considered, by the computer, to be less than unity and if, as the result of an arithmetic manipulation, the absolute magnitude of the results exceeds "computer unity" an overflow is said to have occurred. A programmer must recognize the possibility of this situation and be prepared to cope with it. It should be noted here that overflow will not occur from the use of the "shift" instructions.

It is the purpose of this chapter to discuss the problem of overflow and to present a number of exercises to illustrate the techniques of handling it.

Before entering this discussion, the use of two symbols should be understood. An asterisk, placed in front of a letter indicating the arithmetic operation, suggests the possibility of overflow. Also, the use of brackets [] and (), embracing an instruction, or part of an instruction, indicates that the enclosed quantity will undergo some change in the procedures to follow.

Sec. 2. Overflow Due to Addition or Subtraction

It is apparent that, as the result of an algebraic addition or subtraction of two quantities, the sum or difference may exceed "computer unity" and overflow would occur. The carry digit would be lost and the remaining digits together with the correct sign would remain in rA. The computer reacts to an "overflow" situation automatically. The sequence of instructions is interrupted, and the pair of instructions in memory location 000 is inserted. This insertion is effected after <u>both</u> instructions of the original pair have been executed even though overflow may have been caused by the first instruction of the pair.

If memory location 000 contains an instruction which transfers control, a new instruction sequence is initiated. It is important to note that, if no transfer of control is ordered, the original sequence of instructions is resumed after executing the pair contained in 000.

Example 1: (049) = x, (050) = y. Deliver x + y = z to 0 51.

Mem. Loc.	Instruction		Remarks
Case I			
020	B 049		x --> rA and rX
	*A 050		y --> rX; x+y = z --> rA; if overflow, act on instructions contained in memory location 000.
021	C 051		If no overflow, (rA) = z --> 051;
	00 000		0 --> rA.
Case II			
020	00 000		
	B 049		x --> rA and rX
021	*A 050		y --> rX; x + y = z --> rA; if overflow, $\pm$[|x + y |-1] = z --> rA.
	C 051		(rA) = z --> 051; 0 --> rA; if overflow, act on instructions contained in memory location 000.

Example 2: An addition is performed in a "floating" decimal point routine. If an overflow occurs, (a) the sum must be shifted one position to the right, (b) a "one" must be extracted into the most significant digit position, and (c) the exponent of the sum must be increased by one.

(050) = A
(051) = B
(052) = C
(053) = c = exponent of C
(054) = E_1 = 010000 000000
(055) = -1 000 U 100
(056) = 001000 000000

```
Mem.
Loc.     Instruction          Remarks

020    B  055                 [-1 000  U  100] --> rA
                C  000        (rA) --> 000; set overflow routine
021    B  050                 A --> rA and rX
                *A 051        B --> rX, A+B = C --> rA

- - - - - - - - - - - -

     if overflow

000    -1 000                 SR1(C)
                U  100        obtain next instruction from 100.

- - - - - - - - - - - -

100    F  054                 E1 = 010000  000000 --> rF
                E  054        E1 (E1) --> C
101    C  052                 C --> 052; 0 --> rA
                B  053        c --> rA and rX
102    A  056                 +1 --> rX; c+1 --> rA
                C  053        c+1 --> 053; 0 --> rA
103    B  052                 C --> rA and rX
                U  022        obtain next instruction from 022.

- - - - - - - - - - - -

022    C  052                 C --> 052; 0 --> rA.
```

Instructions which are considered in greater detail in Chapter 8 are used in the following example and, hence, they are defined here also. The instruction "50m" will print one word from memory location m onto the printer associated with Supervisory Control. Furthermore, it will be seen in Chapter 8, that the symbols ℟ and ℐ represent "carriage return" and "ignore" instructions to the printer.

Example 3: If the difference of two quantities overflows (is less than -1) print "overflow" and stop the computation.

```
(050) = A
(051) = B
(052) = C
(053) = [50 054  90 000]
(054) = ℟OVERF   LOWℐℐℐ
```

```
Mem.
Loc.     Instruction        Remarks

020    00 000
                 B  053     [50 054  90 000] --> rA
021    C  000               (rA) --> 000; set overflow routine
                 B  050     A --> rA and rX
022   *S  051               -B --> rX; A -B = C --> rA
                 C  052     (rA) = C --> 052; 0 --> rA

- - - - - - - - - - - -

000    50 054               print "overflow"
                 90 000     stop the computer

- - - - - - - - - - - -
```

Example 4: It is desired to repeat subroutine Σ (memory location 010 through 020) ten times and then proceed to subroutine μ (memory locations 205 through 229).

```
(051) = +1
(052) = n
(053) = [00 000  U  205]
```

```
Mem.
Loc.     Instruction        Remarks

008    00 000
                 B  053     [00 000  U  205] --> rA
009    C  000               (rA) --> 000; set overflow routine
                 C  052     P   > n (052)
010
 .
 .                          Subroutine Σ
 .
020

021    B  052               n --> rX and rA
                *A  051     +1 --> rX; n+1 --> rA

- - - - - - - - - - - -

when n = 10, overflow occurs

000    00 000
                 U  205     go to subroutine μ

- - - - - - - - - - - -

022    C  052               (rA) = n+1 --> 052
                 U  010     go to repeat subroutine Σ
```

Example 5: It is desired to repeat a certain computation (subroutine Σ , memory locations 010 through 020) containing in 018, the pair of instructions Mm Cm for the 484 points (memory locations 516 through 999) of a certain mesh. After the mesh has been swept, it is desired to go to an editing routine starting at memory location 023.

```
                (050) = [M99516  C  516] See Note Below.
                (051) = 000001   000001
                (052) = 00 000   U  023
```

(Note: The computer ignores the second and third digits "99" in an instruction like M99516)

```
008   B  052              [00 000  U  023] --> rA and rX
            C  000        (rA)  --> 000; set overflow routine
009   B  050              [M99516  C  516] --> rA
            C  018        (rA) --> 018; 0 --> rA
010
 .
 .
 .
018  [M(99516)            Subroutine
 .           C (516)]
 .
 .
020

021   B  018              [M99516  C  516] --> rA and rX
          *A  051          000001  000001  --> rX
                           [M99517  C  517]   > rA

- - - - - - - - - - - -

overflow occurs when addition gives [M(00)000  C(01)000]

000   00 000
            U  023        go to editing subroutine

- - - - - - - - - - - -

022   C  018              [M99517  C  517] --> 018
            U  010        return to compute next point

_____________________

023                       start of editing subroutine
```

Example 6: If successive overflows may occur, each requiring a different overflow routine, the following coding device may be employed to avoid the necessity of resetting the overflow instruction contained in 000. Suppose that [R 092 U 089] has been stored in 000.

```
Mem.
Loc.      Instruction        Remarks

010   B   (m1)               (m1) --> rA and rX
              *A  (m2)       (m2) --> rX; (m1) + (m2) -->
                             rA; n = 010.
011
 .
 .                           normal routine
 .
015           U  ---
- - - - - - - - - - - -
016
 .
 .                           overflow routine
 .    U  ---
---
- - - - - - - - - - - -
030   B   (n1)               (n1) --> rA and rX
              *A  (n2)       (n2) --> rX; (n1) + (n2) -->
                             rA; n = 030
031
 .
 .                           normal routine
 .
035
              U  ---
036
 .
 .                           overflow routine
 .
---
              U  ---
```

```
089   B  092                [00 000  U (n+1)] --> rA and rX
               A  204       (000000  000005) = (204) --> rX;
                            (rA) + (rX) --> rA
090   C  092                (rA) = [00 000  U (n+6)] --> 092
               U  092
091   -- ---
               -- ---       Constant
092  [00 000
               U (n+6)]     obtain next instruction from
                            memory location (n+6) for appro-
                            priate overflow routine.
```

Thus, in case of an overflow caused by the addition on line 010,

Mem. Loc.	Instruction		Remarks
010	B (m_1)		(m_1) --> rA and rX
		*A (m_2)	(m_2) --> rX; (m_1) + (m_2) --> rA; overflow
- - - - - - - - - - - -			
000	R 092		[00 000 U 011] --> 092
		U 089	obtain next pair of instructions from 089
089	B 092		[00 000 U 011] --> rA and rX
		A 204	5 --> rX; (rA) + 5 --> rA
090	C 092		[00 000 U 016] --> 092
		U 092	obtain next pair of instructions from 092
092	[00 000		
		U (016)]	obtain next pair of instructions from 016; i.e., start overflow routine

It should be noted that this routine requires neither time nor operations until, and unless, overflow occurs. It is therefore, of greatest value when substituted for such tests as those for end of block, end of mesh, etc.

Example 7: An order is to be constructed to shift left s places where $0 \leq s \leq 10$. The quantity A is to be shifted left s places.

(050) = 01 000 00 000 = SL_1
(051) = B 052 C 024
(052) = 01 000 09 000 = SL_{10}
(053) = A
(054) = s-1

Mem. Loc.	Instruction		Remarks
020	00 000		
		B 051	[B 052 C 024]
021	C 000		--> 000; set overflow routine
		B 050	[01 000 00 000] --> rA
022	*A 054		+ s-1
		C 024	[0s 000 00 000] --> 024
023	00 000		skip; i.e., continue to next instruction
		B 053	A --> rA
024	[00 000		SL_s (A)
		00 000]	
025	C 053		(rA) --> 053; 0 --> rA.

If s = 0, the instructions in 024 become [00 000 00 000].
If s = 5, the instructions in 024 become [05 000 00 000].
If s = 10, overflow occurs, and the instructions become [01 000 09 000].

Sec. 3. Practice Exercises Involving Overflow

1. If the sum of the absolute values of two quantities A and B overflows, stop computation. If not, print "No Overflow" and transfer control to a new routine.

2. Three quantities A, B, C, represented in floating decimal notation, are stored in 050, 051, 052. They are to be added and the sum D is to be placed in 060. Provide the instructions for this routine. Do not stop for overflow but shift appropriately and continue.

3. There are six 10-word input items, located in memory locations 060 to 119, which are to be transferred to memory locations beginning with 240. When this has been done, control is to go to 500. Place proper instructions in 000 and begin routine with memory location 150

```
(150) = [Y94060  Z  240]
(200) =  001010  000010
```

4. Prepare a routine which will evaluate

$$Y = 32.56x^3 - 27.74x^2 + 71.22x + 19.00$$

for 50 values of x from

$x = 1$ to $x = 50$ in intervals of 1.

Sec. 4. Overflow Due to Division

It should be clear that, for all proper divisions $|Dr| > |Dd|$ and $Dr \neq 0$. If an improper division occurs, the sequence of instructions is automatically interrupted, and the pair of instructions in memory location 000 is inserted. This insertion is effected after both instructions of the original instruction pair have been executed, even though overflow may have been caused by the first instruction of the pair.

If memory location 000 contains an instruction which transfers control, a new instruction sequence is initiated. If no transfer is instituted, the original sequence of instructions is resumed after executing the pair in 000.

When overflow due to division occurs, the quantity delivered to rA is $10\,|Dd - Dr|$. The reason for this will be more evident when the reader considers the principles of computer division presented in Chapter 9.

Example 1: In a floating decimal point routine, it is desired to divide N by D to obtain Q (the corresponding exponents are n, d, and q). All quantities are so positioned that their first non-zero digit lies in the most significant digit position. Hence, if an overflow occurs (a) the numerator must be shifted one position to the right and (b) the exponent of the quotient must be increased by one. If a second overflow occurs when the division is repeated, D = 0, the UNIVAC is instructed to print "infinity" and stop.

```
(050) = N
(051) = D
(052) = Q
(053) = q
(054) = [B  050  U  134]
(055) =  001000  000000
(056) = [50 057  90 000]
(057) =  ¤INFIN  ITY¤¤¤
```

Mem. Loc.	Instruction		Remarks
020	B 054		[B 050 U 134]
	C 000		--> 000; set overflow routine No. 1
- - - - - - - - - - - -			
021	L 051		D --> rL and rX
	*D 050		N --> rA; Q = N/D --> rA; (rL) = D.
- - - - - - - - - - - -			
022	C 052		(rA) = Q --> 052
	00 000		
- - - - - - - - - - - -			
Overflow Routine No. 1			
000	B 050		N --> rA and rX
	U 134		obtain next instructions from 134
- - - - - - - - - - - -			
134	-1 000		SR_1 (N)
	C 050		(rA) = N --> 050

```
135   B  053                 q --> rA and rX
           A  055            1 --> rX; q + 1 --> rA
136   C  053                 (rA) = q + 1 --> 053
           B  056            [50 057  90 000] --> rA and rX
137   C  000                 --> 000; set overflow routine No. 2
           U  021            return to repeat division
```

Overflow Routine No. 2

```
000   50 057                 print "infinity" on Supervisory
                                                  Control
           90 000            stop the computation
```

Example 2: In computing a denominated payroll, it is known that the quantity P is such that $20 > P ≥ 0. It lies in the position $0XX_{\Delta}XX0$ 000000 in memory location 050. It is desired to determine whether or not P $10., and if P ≥ $10 to enter a tally in 060 of one ten dollar bill.

(030) = 010000 000000
(031) = 000000 000001
(050) = P
(060) = T_{10} (tally of $10 bills)
(061) = [F 031 G 060]

```
Mem.
Loc.    Instruction          Remarks

020   B  061                 [F 031 G 060] --> rA
           C  000            --> 000; set overflow routine
021   L  030                 .1 --> rL and rX
          *D  050            P --> rA; P/.1 --> rA
```

Overflow Routine

```
000   F  031                 1 --> rF
           G  060            (rF) = 1 --> 060 for tally
```

Note: If overflow does not occur, P is returned to rA in the position, $0X_{\wedge}XX00\ 000000$. If overflow occurs, P - 10 remains in rA in the position $0X_{\wedge}XX00\ \ 000000\ \ (10|P-10|)$ In either case the quantity is ready to be tested for $5 bills.

Sec. 5. Practice Exercises Involving Overflow

1. (050) = 043254000052 = X
   (051) = 025063000052 = Y

Quantities X and Y are represented in floating decimal notation. Provide a routine which will place the quotient $Q = \frac{x}{y}$ in memory location 060.

2. Set up a routine to solve the simultaneous equations

$$a_1 x+b_1 y = c_1$$

$$a_2 x+b_2 y = c_2$$

for x and y.

Assume that equations are independent and that the constants a_1, b_1, c_1, a_2, b_2, c_2, have values ranging between + 10 and - 10.

3. Set up a routine to determine the twenty-five values of z where

$$z_{ij} = \frac{2x_i+3}{3y_j-1} \quad \text{where}$$

x and y each assume 5 values

$$x_i = -2,\ 0,\ +2,\ +4,\ +6$$

$$y_i = -3,\ -1,\ +1,\ +3,\ +5$$

Sec. 6. Instructions to Stop Computer on Overflow

If desired, an instruction can be inserted to stop the computer in case overflow develops and on-the-stop remedial steps can be taken to handle the situation.

Instructions	
A-m D-m S-m X-m	By placing minus signs in the second digital positions of these instructions the computer will stop if overflow develops.

Chapter 8

Input, Output

SEC. I PRELIMINARY DISCUSSION

In the foregoing chapters, the instructions which direct the computer to perform the variety of internal operations necessary to the proper processing of data were described. It is the purpose of this chapter to present the instructions designed to get this information into the computer and those instructions needed to obtain the results from the computer.

For proper understanding of the input and output instructions it is advisable to review the functions of certain component units of UNIVAC, especially as they are related to the current discussion. First, it will be recalled that the UNITYPER is equipped with a standard typewriter keyboard and an auxiliary numeric keyboard. The function of this unit is to convert raw data into a pattern of pulses, with a pulse density of 20 to the inch, recorded on the magnetic tape. The programmer prepares this input information in the form of "words" of data and "words" of instructions -- each word consisting of twelve characters. It is important to note that when an instruction pair is unityped twelve characters must be inserted. For example, if the instructions are B 325 Q1 000, the typist must depress the keys, B00325 Q10000.

The tapes are mounted on UNISERVOs which physically control the tapes in accordance with the instructions contained in the central computer. Any number of UNISERVOs, up to ten (and numbered 1, 2, 3, 4, 5, 6, 7, 8, 9, -) may be used. These UNISERVOs are under automatic control and it should be clear that they may be used as either input or output devices. They can feed data into the computer, "reading" the tape as it moves in either a forward or backward direction. However, in recording the desired output, the "writing" is performed only with the tape moving in a forward direction.

The Supervisory Control is a main console keyboard, and contains a typewriter dolly, all manual control keys and indicator lights. The operation "initial read", performed by depressing a switch on Supervisory Control, transfers the first block (120 instructions) from the magnetic tape on the appropriate UNISERVO to memory locations 000----059.

Thereafter, the computer automatically "reads" from magnetic tape to its memory, or conversely "writes" from memory onto magnetic tape according to the instructions set up by the programmer. Information is transferred between the tapes on the UNISERVOs and the computer in sixty word blocks. Moreover, if proper instructions are inserted into the computer, the results may be printed on the electric typewriter associated with the Supervisory Control

The UNIPRINTER contains an electric typewriter (and a tape reading unit) which translates the magnetic impulses on the tape into printed copy. All keys, including upper and lower cases of the alphabet, punctuation marks, spaces, tabs and carriage returns, operate automatically. However, margin and tab stops are set by hand. A tape "edited" by the UNIVAC for printing is complete in all details; for example, with proper coding, zeros to the left of significant digits are suppressed, tabs are interspersed and carriage returns operate at the end of each printed line.

It is, also, advisable to review, briefly, the functions of rI and r0 as they are related to the input-output processes.

Register I: Register I is a sixty word register. Information transferred to the computer is read from magnetic tape into rI, and then, from rI into the memory.

Register I is unlike other UNIVAC registers because it does not erase on read-in but only on read-out. Therefore, rI must be erased before information is transferred from tape to rI. If rI is not erased, a read causes new digits to be superimposed upon those already in rI. This usually results in an error signal when the contents of rI are transferred to the memory, since the resulting combinations very probably will fail to satisfy the odd-checking system.

When a backward read instruction is executed, rI receives the information from the tape in reverse order. To maintain the same relative order within the block, the first digit from the tape is stored in the least significant digit position of rI. The digits from the tape are then filled consecutively in a backward direction. The net result is

that all digits in rI appear in the same order whether a backward or a forward read instruction has been executed.

Register 0: Sixty words are transferred from the memory to r0, thence to tape, during a write instruction. Unlike rI there is no separate transfer instruction to or from r0. Therefore, r0 need not be considered in coding.

SEC. 2. FORWARD READ

INSTRUCTIONS

1nm	Read one block of data (60 words) from the tape mounted on UNISERVO n to rI. The tape moves in a forward direction. The memory location, m, is ignored.
3nm	Transfer one block of data from rI to sixty consecutive memory locations beginning with m, and erase rI. Then, read one block of data from the tape mounted on UNISERVO n to rI. The tapes move in a forward direction. The least significant digit of m is ignored by the computer, i.e., m is treated as an integral multiple of 10.
30m	Transfer one block of data from rI to sixty consecutive memory locations beginning with m, and erase rI. The least significant digit of m is ignored by the computer.

To complete a transfer from tape to memory, it is necessary to read from tape to rI and from rI to memory. Note: rI must be erased before information is transferred from tape to rI.

Example 1: Initial read has been performed, and rI is erased. Read one block of information from tape 1 to memory locations 060...119 and then erase rI.

Mem. Loc.	Instruction	Remarks
000	11 000	Tape 1 --> rI
	30 060	rI --> 060...119; erase rI

Example 2: Initial read has been performed and rI is erased. Read two blocks of instructions from tape 1 to memory locations 060...179. Read two blocks of data from tape 2 to memory locations 400...459 and 200...259. Read one block of data from tape 10 to memory locations 300...359, leaving the next block of data from tape 10 in rI.

Mem. Loc.	Instruction	Remarks
000	11 000	Tape 1 --> rI
	31 060	rI -->060...119; Tape 1 --> rI
001	32 120	rI -->120...179; Tape 2 --> rI
	32 400	rI -->400...459; Tape 2 --> rI
002	3- 200	rI -->200...259; Tape 10 --> rI
	3- 300	rI -->300...359; Tape 10 --> rI

SEC. 3 BACKWARD READ

INSTRUCTIONS

2nm	Read one block of data (60 words) from the tape mounted on UNISERVO n to rI. The tape moves in a backward direction. The memory location, m, is ignored.
4nm	Transfer one block of data from rI to sixty consecutive memory locations beginning with m, and erase rI. Then, read one block of data from the tape mounted on UNISERVO n to rI. The tape moves in a backward direction. The least significant digit of m is ignored by the computer i.e., m is treated as an integral multiple of 10.
40m	Transfer one block of data from rI to sixty consecutive memory locations beginning with m, and erase rI. The least significant digit of m is ignored by the computer. (Note that 30m and 40m instructions are identical).

A backward read cannot be used to start a problem since all tapes must be mounted on the left hand reel of the UNISERVO. Tapes move from the left reel to the right reel during a forward read, and in the opposite direction during a backward read.

A backward read is accomplished by moving the tape in a backward direction and transferring digits from tape to rI in inverse order. For this reason information is transferred from rI to memory and in the same relative order as for a forward read.

Example 1: Assume rI is erased. Read one block of information from tape 1 to memory locations 200...259 with tape 1 moving in a backward direction. Erase rI.

Mem. Loc.	Instruction	Remarks
100	21 000	rI <-- tape 1
	40 200	rI --> 200...259; erase rI

Example 2: Transfer the contents of rI to memory locations 200...259. Read two blocks of data from tape 2 to memory locations 260...319 and 320...379 with the tape moving in a backward direction. Read one block of data from tape 10 to memory locations 400...459 with the tape moving in a backward direction. Erase rI.

Mem. Loc.	Instruction	Remarks
100	42 200	rI --> 200...259; rI <-- tape 2
	42 260	rI --> 260...319; rI <-- tape 2
101	4- 320	rI --> 320...379; rI <-- tape 10
	40 400	rI --> 400...459

SEC. 4. PRACTICE EXERCISES ON FORWARD AND BACKWARD READS

1) An input tape, which contains four blocks of information is mounted on UNISERVO 1. The initial read has been performed. If the tape is moving in a forward direction, provide the instructions to send blocks 2, 3, 4, to memory locations 60... 119, 120...179, 180...239, clear rI.

2) Provide the instructions needed to do the following: transfer the contents of rI to memory locations 320 to 379. Read two blocks of data from tape 10, moving in a forward direction to memory locations 100 to 159 and 160 to 219. Also, read two blocks of data from tape 5, moving in a backward direction. Send the first block of locations 500-559 and leave the second block in rI.

3) Initial read has been performed on tape 1. Provide the coding necessary to transfer one block of information from each of tapes 2, 4, 6, 8, to memory locations beginning with 200, 400, 600, 800 respectively. Tapes 2 and 6 are moving in a forward direction; tapes 4 and 6 are moving in a backward direction. Clear rI.

SEC. 5 WRITE INSTRUCTIONS

Writing on tape may be done in only one direction, forward. If a tape already contains information and a write is executed on block n of the tape, the information in the succeeding blocks n + 1, n + 2, n + 3, etc. cannot be read. The write mechanism does not always space blocks in exactly the same physical position on the tape. Block n, which has just been written, may have been partially superimposed on block n + 1. This means that the following blocks are out of register.

There are two write instructions available: one writes on tape at 100 pulses/inch, the other writes on tape at 20 pulses/inch. Which of the two is used depends on the future operations to be performed on the reel in question. All tapes to be Uniprinted must be written at 20 pulses/inch. This is the only pulse density acceptable to the UNIPRINTER. All tapes to be used again in UNIVAC operations should be written at 100 pulses/inch. The UNIVAC will accept either pulse density; however, tape at 100 pulses/inch may be read approximately five times as fast as a tape at 20 pulses/inch.

INSTRUCTIONS

5nm	Write one block of data, from the sixty consecutive memory locations beginning with m, on the tape mounted on UNISERVO n. The tape moves in a forward direction. The least significant digit of m is ignored by the computer. The pulse density on the tape is 100/inch. This tape may be used only in future UNIVAC operations.
7nm	Write one block of data, from the sixty consecutive memory locations beginning with m, on the tape mounted on UNISERVO n. The tape moves in a forward direction. The least significant digit of m is ignored by the computer. The pulse density on the tape is 20/inch. This tape may be used in future UNIVAC or UNIPRINTER operations.

Example 1: Write two blocks of data from memory locations 200...259 and 260...319 on tape 6. Write one block from memory locations 400...459 on tape 10. Tape 6 is to be used in future UNIVAC operations. Tape 10 is to be Uniprinted.

Mem. Loc.	Instructions		Remarks
100	56 200		200...259 --> tape 6
		56 260	260...319 --> tape 6
101	7- 400		400...459 --> tape 10
		00 000	Skip

SEC. 6. REWIND INSTRUCTIONS

Rewind instructions are used to return the tapes to the left hand reel of the UNISERVO after they have been processed. This is necessary because tapes may be removed only from the left hand reel.

One or two tapes may be rewound simultaneously. If a third rewind instruction is given during the time two tapes are already rewinding, the computer will interlock until one of the two previous rewinds has been completed.

INSTRUCTIONS

6nm	Rewind the tapes mounted on UNISERVO n to the beginning. The memory location m is ignored by the computer.
8nm	Rewind the tape mounted on UNISERVO n to the beginning. This instruction interlocks UNISERVO n and produces a visual signal. No data can be read from or written on the tape associated with UNISERVO n until the manual interlock release switch on this UNISERVO has been actuated. Changing a tape causes this switch to be activated.

Example 1: Rewind the tape mounted on UNISERVO 4 with interlock. Rewind the tape mounted on UNISERVO 5 without interlock.

Mem. Loc.	Instruction	Remarks
100	84 000	Rewind tape 4 with interlock
	65 000	Rewind tape 5

SEC. 7. INTERLOCK

If a read (write) instruction is being executed and another read (write) is ordered, computation is interrupted. Under this set of conditions the computer is said to be interlocked. This interlocking condition will remain until the first read (write) instruction is completed. Then the computer automatically resumes operation, and executes the instruction which caused the interlock.

SEC. 8 SUPERVISORY CONTROL

The programmer may arrange to use the Supervisory Control as an input-output device. When this is done certain options are available depending on the switch settings. These options are explained in "Supervisory Control Operations". The two programmed instructions which may be used for input or output in connection with the Supervisory Control are listed.

INSTRUCTIONS

10m	Stop UNIVAC operations and produce a visual signal. Call for one word to be typed from the Supervisory Control keyboard into memory location m. UNIVAC operations are resumed after the "word release button" on the Supervisory Control has been actuated.
50m	Print one word from memory location, m, onto the printer associated with the Supervisory Control. UNIVAC operations are resumed automatically after m has been transferred to an intermediate output storate location prior to printing.

The ability to type into or print out of any desired memory location during the processing of a problem permits a very flexible control of that problem. However, it is well for the programmer to remember that the time required to execute these instructions is relatively great especially for a type-in instruction which is a human operation and an added source of error.

SEC. 9. PRACTICE EXERCISES

Problems on write, rewind, Supervisory Control orders.

1) Provide the coding needed to write three blocks of data from memory locations 300...359, 360...419, 420...479 on tape 10. Write two blocks from memory locations 500...559, 600...659 on tape 3. Tape 10 is to be used in future UNIVAC operations and tape 3 is to be Uniprinted. Also, rewind tapes 3 and 10.

2) The results to a computer problem have been organized into five blocks of data and are located in memory locations 500 to 559, 560 to 619, 620 to 679, 680 to 739, 740 to 799. Provide the instructions which will print the first word of each block on Supervisory Control printer and also which sends the first three blocks to tape 3 and the fourth and fifth blocks to tape 6. All information is to be written for future Uniprinting; also tapes should be rewound.

3) Four blocks of data are located on tape 10 and are to be processed through the computer. Assume that initial read has been performed. Provide the instruction which will send these four blocks of data to memory locations 200 to 299, 300 to 307, 400 to 459, 500 to 559 and clear rI. Then introduce the 10m instructions which will enable the operator to replace the data in the first word of each block. Also rewind the tape.

SEC. 10. EDITING

General Description

Editing may be classified according to its use as input editing or output editing. Input editing is essentially a combination of shift and extract operations which rearrange fields in order to facilitate computation. Output editing is

similar to input editing in the operations which are used. However, since its purpose is to "edit" the output so that when it is printed, the information will be easy to read and understand, it also includes (1) the insertion of standard punctuation and symbols such as commas, decimal points, periods and percent signs, and (2) the insertion of printer operation instructions such as carriage return, tab, space, shift lock, and printer stop.

UNIPRINTER Instructions

The only instructions not previously described are the UNIPRINTER instructions. These differ in that each digit is essentially an instruction. The coding symbols for the UNIPRINTER instructions are given in the first column of the table that follows. The UNIPRINTER can be set to operate on either "Normal" or "Computer Digit". If the printer is set on "Normal", it will perform the operations shown in column two. If the printer is set on "Computer Digit", it will not perform the operations indicated with two exceptions; it will "Space" and "Stop" when called for. For the other instructions the printer will type out the symbols given in column three.

UNIPRINTER Instruction	Normal Position	Computer Digit
R	Carriage Return	Types /
T	Tab	Types *
L	Shift Lock	Types z
U	Shift Unlock	Types 8
Ø	Single Shift	Types -
Δ	Space	Space
I	Ignore	Types x
B	Printer Breakpoint	Types y
P	Printer Stop	Printer Stop

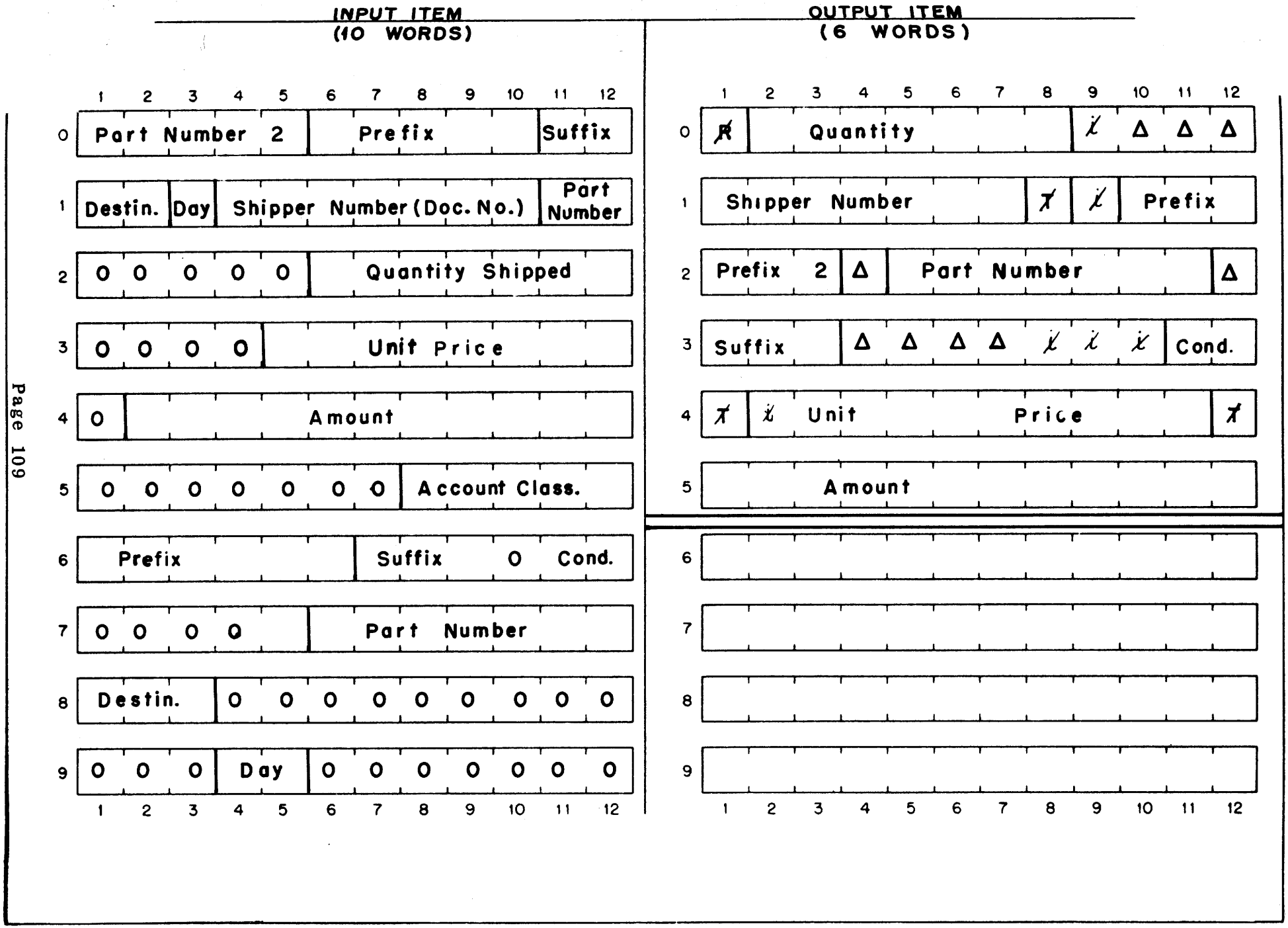
Use of Lay-Out Sheets as an Assist in Editing

Every problem in editing is different. There may be some short subroutines which can be standardized but in general, each problem must be approached from the standpoint of what is to be accomplished during a particular run or group of runs. The lay-out sheet is a great aid in the analysis of the problem as well as to the coding and subsequent checking.

Before the coding of the problem is started, the input item and the output item including the heading for the output page should be put on the lay-out sheets. This "would-be" heading and output item is then tested on a standard typewriter. At this stage of the problem, any desired change can be made quite easily. Also, if any similar types of editing exist, their presence is seen more easily. When this point in the editing problem is reached, the major part has been completed. The programmer knows what he has, what he wants and the form it is to take. The remainder is routine coding.

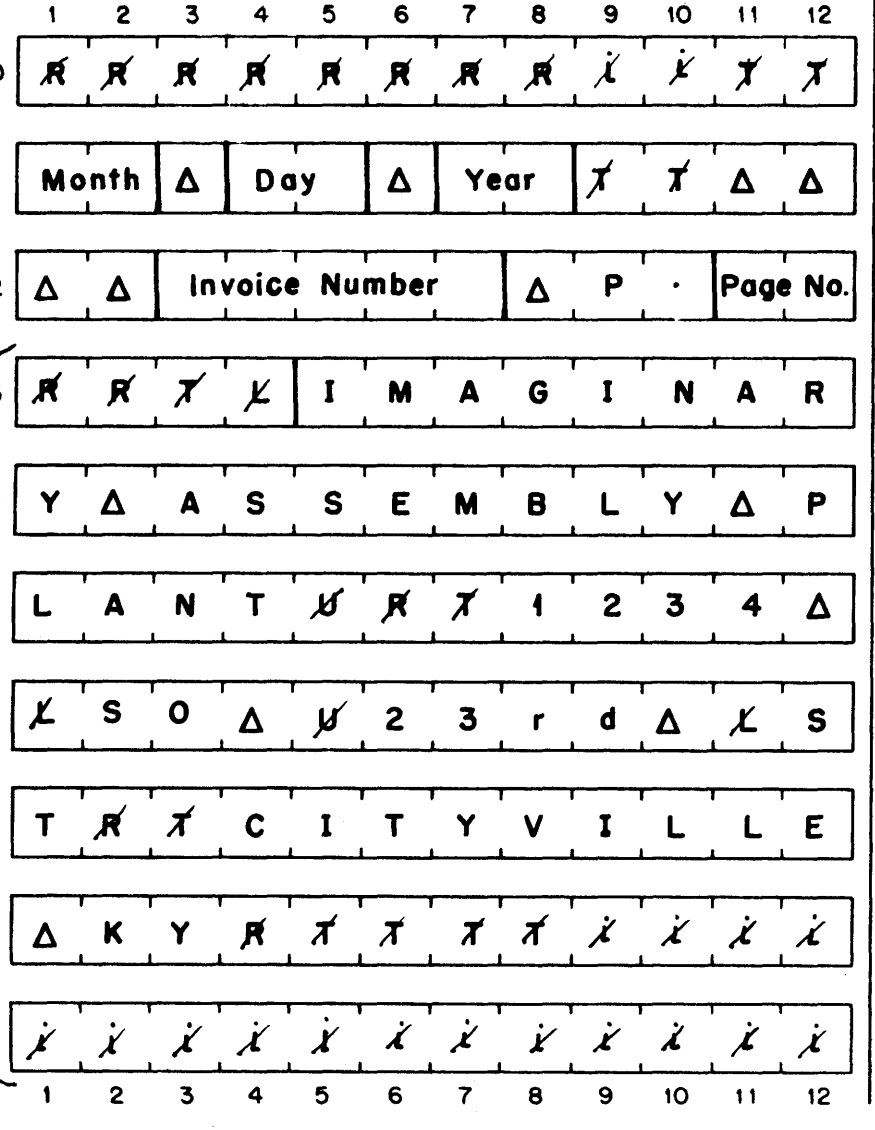
Examples of Editing

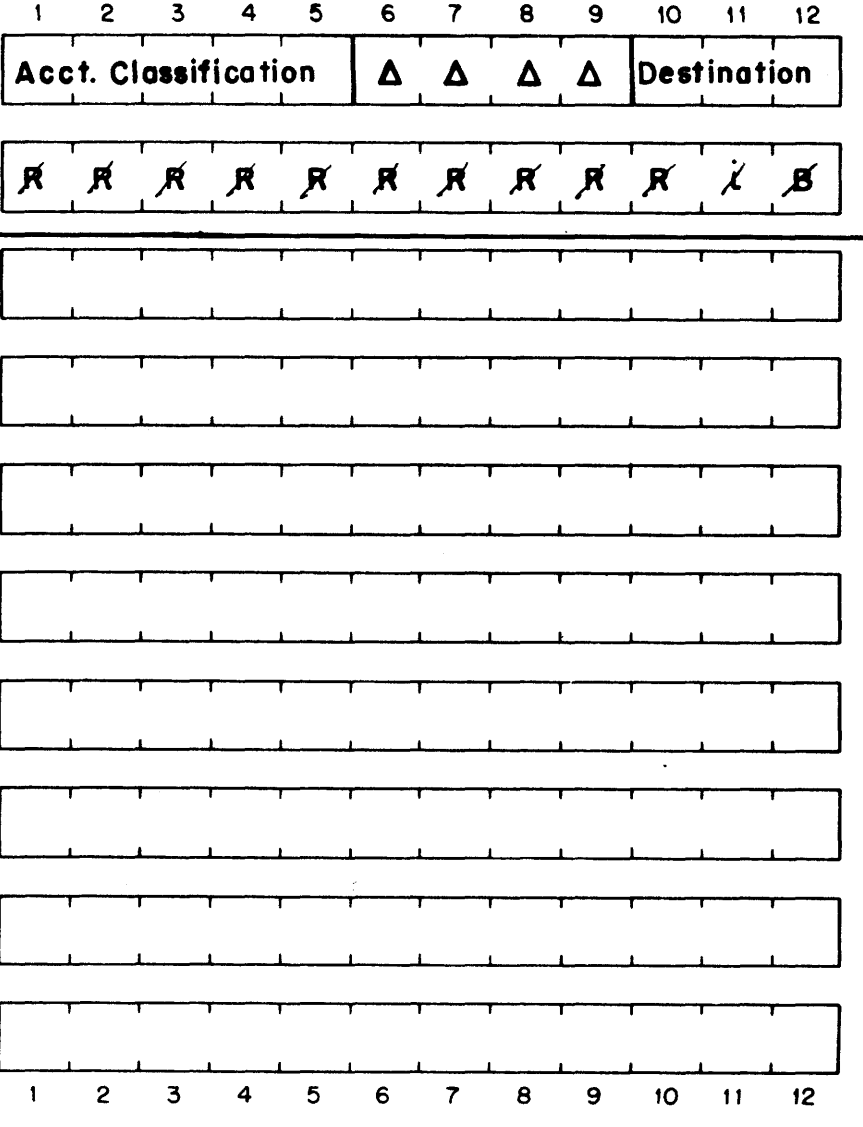
Two lay-out sheets are included on the next two pages in order to clarify some of the examples which will be given. It will be noted that the input item is a ten-word item with an apparent waste of space. This was done to facilitate the handling within the computer. It will also be noted that the output item is only a 6-word item and is rather closely packed. The heading output is a 12-word item (multiple of 6). The output will not be used again in the computer but will go to the UNIPRINTER where the amount of material to be printed is the main consideration. The length of the output item is usually a factor of 60.


INPUT ITEM
(10 WORDS)
OUTPUT ITEM
(6 WORDS)
1 2 3 4 5 6 7 8 9 10 11 12
0 Part Number 2 Prefix Suffix
1 Destin. Day Shipper Number (Doc. No.) Part Number
2 0 0 0 0 0 Quantity Shipped
3 0 0 0 0 Unit Price
4 0 Amount
5 0 0 0 0 0 0 0 Account Class.
6 Prefix Suffix 0 Cond.
7 0 0 0 0 Part Number
8 Destin. 0 0 0 0 0 0 0 0 0
9 0 0 0 Day 0 0 0 0 0 0 0
0 Quantity
1 Shipper Number Prefix
2 Prefix 2 Part Number
3 Suffix Cond.
4 Unit Price
5 Amount
6
7
8
9
Page 109

OUTPUT ITEM (12 WORDS)

HEADING ITEM WITH ADDRESS





Rearrangement of Fields for Sorting

Assume that an earlier run was sorted by destination, day, shipper number, and part number. For this, the destination (word 08) was reduced to a 2-digit field and the day (word 09) was reduced to a 1-digit field by assigning alphabetic characters to 2-digit days. These fields were then extracted into the position as shown in word 01.

Rearrangement of Fields for Computation

In this particular problem, since the quantity shipped may be as large as a 7-digit field, and the unit price range is an 8-digit field correct to the fifth decimal place, it is impossible to shift the factors so that the amount will be rounded off to cents by 11-place multiplication. The amount, correct to cents, under the above conditions would require a 12-digit field if it were not known that the combination of a large quantity with a high unit price does not occur.

If, in the above problem, the quantity had required no more than a 6-digit field, the quantity and the unit price could have been placed as follows:

```
(02)   0[XXXXXX]    000 0      quantity shipped
(03)   0[XXXXXXXXX]000         unit price
```

so that the 11-place multiplication could be carried out without resorting to 22-place multiplication, adding a round-off and then shifting. This would definitely have simplified the problem. In such situations the chances of the quantity exceeding 999,999 should be investigated; the problem need not be more complicated than necessary.

Address-Type of Input Editing

Consider for a moment the UNIPRINTER instructions necessary to correctly print an address, and the variations in the character and length of various parts. It will be obvious that many complications arise. For this reason, addresses are most often pre-edited --- they are edited as they go on tape.

For example, 7 words have been allotted to the address part as seen in the heading output item on page 110. The initial carriage returns together with the tab set, place the start of the address correctly. Then typed it will appear as

IMAGINARY ASSEMLY PLANT

1234 SO. 23rd St

CITYVILLE KY

The words allotted to an address are seldom manipulated except as a whole.

Insertion of UNIPRINTER Operations

Attention is called to word 00 of the regular output item on page 110. In addition to the UNIPRINTER instructions as shown, it is desirable to have Δs precede the first significant digit of the "quantity". This could have been accomplished in the following manner.

```
     B   XX2                    Quantity from input item position
               00 000                                  --> rA
     R   105
               U  100           --> Zero Suppression Routine


     ;4 000                     SL 4
               F  420           100000001111
     E   421                    Extract Edit Symbols
               C  XX0           --> output item


100  C   483                   00000XXXXXXX --> ws
               B  422           00001-------
101  L   483
               00 000
102  .1 000                    [00001--------] --> ws
               T  102
103  C   485
               B  423
104 [F   485
               E  483]
```

```
105     [00 0000
                        U     ]      Return Line
_______________________________

420      100000        001111        483 w.s. for amount to be edited

421      100000        001ΔΔΔ        485 w.s. for extractor

422      00001-        ------

423      ΔΔΔΔΔΔ        ΔΔΔΔΔΔ
```

It is to be noted that the sub-routine has been coded in such a manner that lines 100-105 may be used for suppressing the zeros in the "part number" (word 07) which will be a part of the output word 02.

<u>Insertion of Punctuation and Standard Symbols</u>

It is desired to suppress the zeros preceding the first significant digit in the "amount" and to insert the decimal point. This can be done in a manner similar to the straight suppression of zeros. With the amount in rA,

```
         R  113      U  106
----------------------------
         C  XX5                  --> Output
----------------------------
106      F  424                  100000 000011
                     H  486      Amount --> working storage
107      E  425                  Extract decimal point and zero
                     F  426      000000 000011
108      ;1 000                  SL1
                     E  486      Extract cents
109      C  487                  Amount with .XX --> w.s.
                     B  427      1-----------
110      L  487
                     T  114
111      C  488                  Fabricated extractor --> w.s.
                     B  423
```

```
112     F  488
                      E  487      Extract amount
113    [00 000
                      U     ]     Return line
-------------------------
114     .1 000
                      U  110
-------------------------
423     ΔΔΔΔΔΔ       ΔΔΔΔΔΔ

424     100000        000011

425     000000        0000.0

426     000000        000011

427     1-----        ------

428

486    [0XXXXX       XXXXXX]     Unedited amount

487    [XXXXXX       XXX.XX]     Partially edited amount

488    [1-----       ------]     Fabricated extractor
```

Chapter 9

Elementary Description of the Operation of a Computer

SEC. 1 PRELIMINARY DISCUSSION

The purpose of this chapter is to present information which will lead to an understanding of the fundamental operation of UNIVAC. It should be clear that the scope of this development will be on an elementary level. In pursuance of this idea it is planned to discuss in some detail a modified simple computer which uses components contained in UNIVAC but, it must be emphasized, this computer will not be UNIVAC. This chapter will be concluded by pointing out the similarities and differences that exist between UNIVAC and the computer herein developed. However, before describing the computer and its relationship to the operation of UNIVAC, it is advisable to review some elementary concepts of electricity.

SEC. 2 BASIC ELECTRICAL CONCEPTS

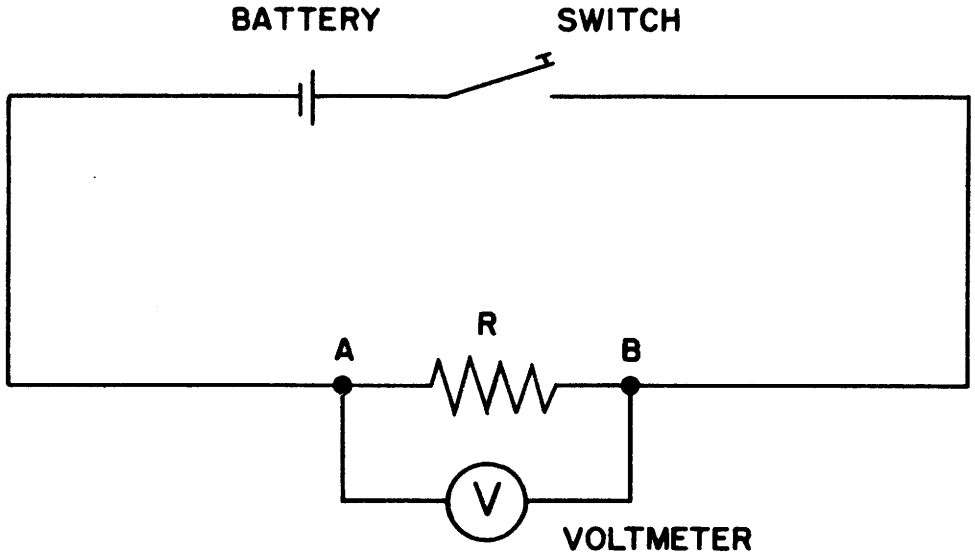


Fig. 1

The simplest form of electrical circuit is a battery with a resistence connected to its terminals as shown in Fig. 1. A complete circuit must have an unbroken path so that current can flow out of the battery, through the elements connected to it and back into the battery. The circuit

is broken or opened if some part of this path is removed. A switch is a device for opening or closing the circuit and, hence, for preventing or allowing current to flow.

The battery in this circuit maintains a difference of potential energy, V (measured in volts), between its terminals which will force current to flow through the circuit elements. The current, I, is defined as the movement of electricity along a conductor and is measured in amperes. It has been found that the current flowing in a circuit is proportional to the difference of potential established between the terminals of the battery. Then $V \propto I$.

This can be written as an equation by introducing a constant of proportionality R. The constant R (measured in ohms) is called the resistance of the circuit and measures the resistance to the flow of current through any given circuit. The expression for difference of potential between two points can be written as $V_{(volts)} = I_{(amps)} \times R_{(ohms)}$ and this relationship is called Ohm's Law.

Thus in Fig. 1, the voltage difference between points A and B can be determined by Ohm's Law if the current in amperes flowing between A and B and the resistance in ohms of this part of the circuit are known, or the voltage difference between points A and B can be measured directly by connecting a device called a voltmeter as shown in Fig. 3.

Suppose that, when the switch is closed, the battery forces a current flow of 2 amperes and that the resistance R is known to be 5 ohms. Then by Ohm's Law it will be known that the voltmeter will register 2 amp x 5 ohm = 10 volts. However, if the switch is opened the circuit is broken and no current flows. The difference of voltage between points A and B would now be 0 amp x 5 ohm = 0 volts.

Thus, by opening and closing the switch, the voltmeter can be made to register either zero or ten volts. Because the battery produces a constant potential difference and the resistance, R, does not vary the voltmeter can assume only one of these two values depending upon the position of the switch.

It is possible to hold point A at a constant potential level, and use this as a reference level. If this is chosen as zero, reference can be made directly to the voltage at point B. For, if the switch is closed, the difference of potential between A and B is ten volts. But since A is fixed at zero then point B must be at ten volts. Considering the circuit in Fig. 1, the voltage of point B will fluctuate between zero and ten volts as the switch is open or closed.

SEC. 3. REPRESENTATION OF INFORMATION

The two possible voltage levels of point B provide a means of representing information. For example, the ten volt condition might represent true, the zero condition false, or the choice yes or no, or the number one or zero. It would seem, however, that the type and quantity of information that can be represented in this manner is limited. This is certainly true if point B is examined at only one instant of time. However, if point B is sampled one second after some reference time and then two seconds after the reference time, making the restriction that the person operating the switch cannot change its position more than once per second, four conditions can be represented.

Referring to Fig. 2, showing a graph of the voltage of point B plotted against time in seconds, we see that point B can have the value 0 volts during the first second, 0 volts during the second second, or ten volts during the first and zero during the second and so on. Four distinct patterns can be used to represent the decimal number 0, 1, 2, 3.

Sampling each of three successive time periods would produce one of 2^3 or eight possible patterns representing the decimal numbers zero through seven. By sampling successive time periods one of 2^n possible patterns would exit. Thus, by choosing the number of time periods sufficiently large, any decimal number can be represented.

It is important to realize that the pattern of voltages of point B representing a number does not appear instantaneously but appears serially. This is necessary because point B can have only one of two possible voltages at any instant of time.

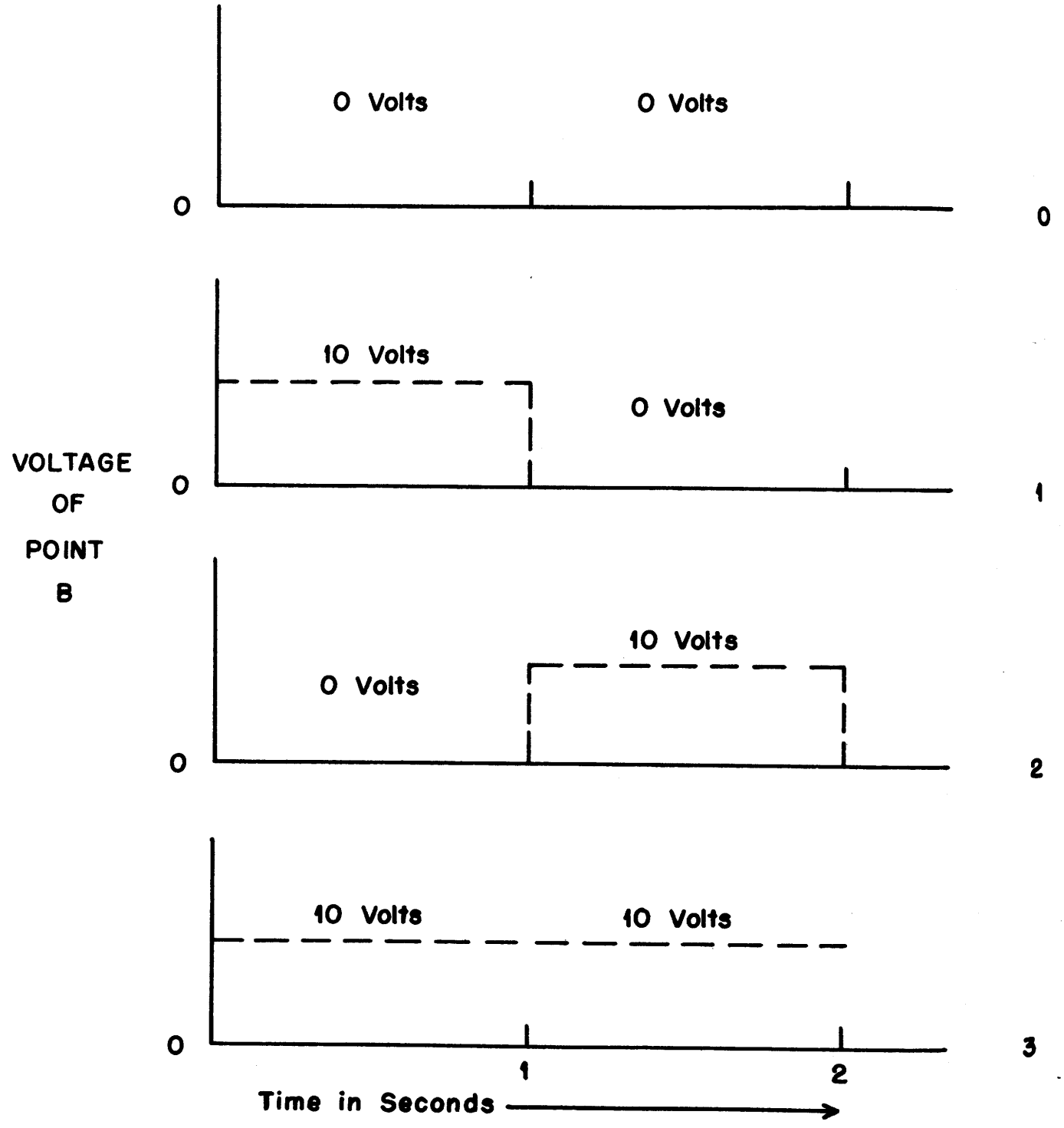

0 Volts
0 Volts
0
0
10 Volts
0 Volts
1
1
VOLTAGE
OF
POINT
B
0 Volts
10 Volts
0
2
2
10 Volts
10 Volts
0
3
1
2
Time in Seconds

FIG. 2

It is advantageous at this point to repeat the structure of the decimal number system as given in Chapter 2. It is called the decimal system because it uses the ten integers 0, 1, 2 ... 9. Furthermore, a number such as 239 is really a short hand method of writing 2 hundreds plus 3 tens plus 9 units. This may also be written as $2 X(10^2) + 3 X(10^1) + 9 X(10^0)$, recalling that $10^0 = 1$, $10^1 = 10$, $10^2 = 10 \times 10 = 100$, $10^3 = 10 \times 10 \times 10 = 1000$, etc. The digits 2, 3 and 9 are really the coefficients of three different powers of ten which is called the base of the system. Again 2,134 is really a short way of writing $2 X(10^3) + 1 X(10^2) + 3 X(10^1) + 4 X(10^0)$.

There is no reason why ten must be the only base that can be used. Examine briefly the binary number system which uses only two integers 0 and 1. Numbers can be expressed in the same general form as in the decimal system. Thus 1101 in the binary system is really a short hand notation for writing.

$$1 X (2^3) + 1 X(2^2) + 0 X(2^1) + 1 X(2^0).$$

Its decimal equivalent would be

$$8 + 4 + 0 + 1 = 13.$$

To represent the number expressed decimally as 17 would require one sixteen and one unit and would be written in binary as

$1 X(2^4) + 0 X (2^3) + 0 X (2^2) + 0 X (2^1) + 1 X (2^0)$ or 10001.

It is interesting to note the similarity in the short hand notation for the binary representation of numbers and the voltage patterns representing numbers in the circuit in Fig. 2. In the binary system numbers are represented by a serial pattern of zeros and ones, and in the circuits by a serial pattern of no voltage and voltage. Thus, there is a direct correspondence between an easily produced characteristic of an electrical circuit and the binary system of representing numbers. This and the fact that the rules of binary arithmetic are very simple make it practicable to use simple voltage patterns to represent information for computational purposes.

Using the circuit described and the time interval of one second to represent a binary digit, it would take ten seconds to represent ten binary digits. Since such a pattern can represent only the range of numbers expressed decimally as 0 ... to 1023, it is evident that such a serial representation of information is quite time consuming.

The obvious remedy would be to decrease the time required to represent a binary digit and hence reduce the overall time to represent a given amount of information. If the time interval mentioned above is reduced from one second to 1/100 of a second, any ten binary digit number can be represented in 1/10 second instead of ten seconds.

It is impossible, however, to manually operate a switch this rapidly or to observe voltmeter readings changing so frequently. It is possible to replace the switch with an electrical device which will open or close the circuit rapidly for accurately measured time intervals. With such a rapidly operating device, voltmeter readings would be impossible to observe but interest exists generally in operating additional electric circuits with the voltage pattern produced and not in directly observing information so represented.

The number of binary digits that can be represented per second is called the frequency. In the initial consideration of the circuit in Fig. 1, it was assumed that the voltage at point B could change only once per second. A binary digit, therefore, lasts for one second; hence, the frequency would be one. In reducing the time required to represent information, it was assumed that the voltage at point B could change once per 1/100 second. Thus, the frequency would be 100, since 100 binary digits could be represented in one second.

The period of time required to represent a binary digit ($= \frac{1}{\text{frequency}}$) is called a pulse time. A pulse is said to be present if a binary one is represented during a particular pulse time. Thus, the binary number 1011 would be represented electrically, least significant digit first, as pulse,

pulse, no pulse, pulse. This is shown graphically in Fig. 3 at a frequency of 100 pulses per second. When denoted in this manner, frequency is sometimes called the pulse repetition rate, or the "rep rate".

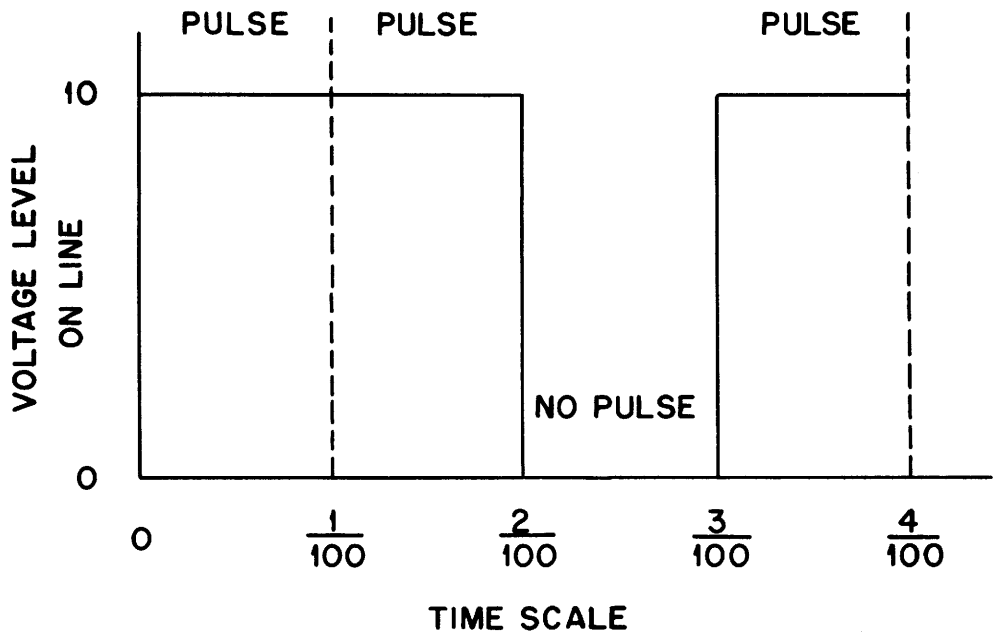


Fig. 3

Voltage changes which persist for relatively long periods of time are called signals. In general, signals will be used for control purposes while pulse patterns are used to represent information.

SEC. 4. CHARACTERISTICS OF A SIMPLE COMPUTER

We are now ready to define the characteristics of the computer mentioned in the first paragraph of this chapter. The reader is again reminded that the description to follow is not that of the UNIVAC, but nevertheless helpful in the understanding of the operation of UNIVAC.

In the simple computer to be considered, the "rep rate" is assumed to be 1,000,000 pulses per second or a frequency of 1 megacycle, (1,000,000 cycles per second). The time required to generate a single pulse or to represent a single binary digit will be 1/1,000,000 second or 1 microsecond (1 μs). It shall be assumed, further, that thirty binary digits will be sufficient to represent the range of numbers that will be used. In order to standardize the timing of the computer, every number shall be represented by thirty binary digits whether or not all thirty positions are necessary. This basic unit of information is called a computer word or word. It has been shown that only one binary position can be represented at one instant of time; binary one with a high voltage or presence of a pulse or binary zero with a low voltage or absence of a pulse. Hence, the entire word must be represented by a serial pattern of pulses. It is possible to represent the word (with respect to time) with the most significant digit (MSD) first or with the least significant digit (LSD) first. To facilitate arithmetic operation the least significant digit will be represented first followed by succeeding digits in order of significance up to the most significant digit.

An additional pulse position will be added after the MSD to indicate the sign of the quantity. No pulse in the sign position will indicate a negative quantity; a pulse in this position will indicate a positive quantity.

In order to allow a fixed interval of time between successive words on a line, additional pulse positions will be assigned which will never contain pulses. There will be seven such pulse positions before the LSD and four following the sign position and will be called the space between words (SBW).

The final structure of the word consists of 42 pulse positions. The first of the seven pulse positions preceeding (time wise) the LSD will be designated as p_0. The remaining positions are numbered consecutively through p_{41} which is the last of the four pulse positions of the SBW following the sign position. This structure is illustrated in Fig. 4.

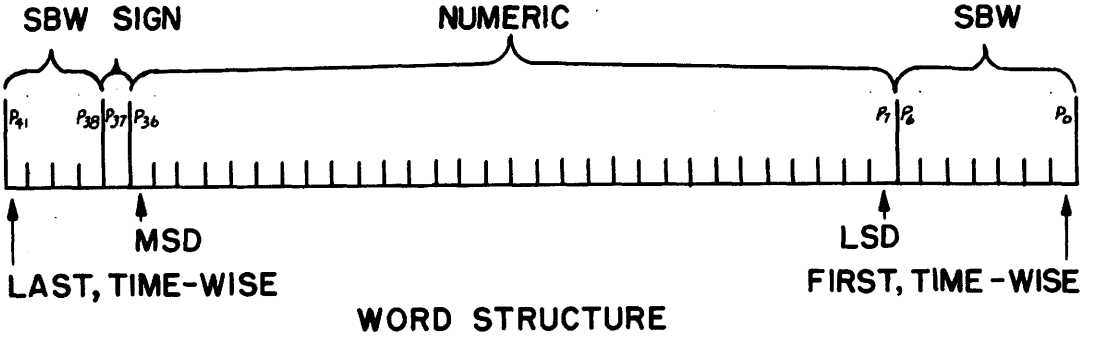


Fig. 4

The basic pulse frequency of 1 Mg. fixes the time to represent one pulse at 1 μsec. Since the word contains 42 pulse positions, the time to represent one word, or the word time, is 42 microseconds.

SEC. 5. COMPONENT UNITS OF THE COMPUTER

At this point, descriptions of certain component units and certain operations of the computer will be introduced. Specifically, the discussion below will attempt to describe the operations of flip-flops, gating and buffing, the comparator, delay mechanisms, adders, complementing, counting, registers, shifting, distribution and collection, function tables.

DYNAMIC VERSUS STATIC SIGNALS AND STORAGE

A radio message may be thought of as dynamic. During the reception of information some of it has already been given, some of it is immediately available, while still more is yet to arrive. In contrast, the message once received and written down is static: the message may be examined in whole or in part at any time after reception.

If the radio message were to be repeated continuously, then the information contained in the message would be dynamically remembered. Any one desiring the information would merely tune in at any time and listen once during through the entire message. (Obviously, the message would only be received in its correct arrangement if the listener tuned in just at the beginning.)

Much of the information in the computer is in dynamic form; it circulates around loops containing electrical delays. Such information can be observed serially as it passes some point in the path, yet all the information is not available simultaneously.

It contrast there are static storage devices in the computer which retain information in static form. All the information stored in static form is immediately available, but it may be realized only by sampling or probing the static device with some signal in order to learn its present state.

One of the most common forms of static memory is the flip-flop (FF). It is indicated on block diagrams by the symbol:

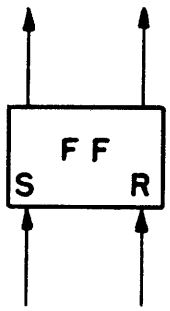


Fig. 5

It is a memory for one binary digit; it has two stable states, one representing zero and the other representing one. These two states are indicated by S (= set) and R (= restore). Its use is sufficiently broad that the binary notation is not always appropriate. For example, it can, in a binary computer, store the sign digit, that is, either a + (which is equivalent to a one) or a - (which is equivalent to a zero). The flip-flop is also useful for converting from a pulse to a static signal. For example, one pulse may indicate when a static signal is to start and another pulse when it is to stop. The FF can be used for generating such a static signal. The duration of the signal is fixed by the interval between pulses.

When a pulse is applied to the "set" input, the FF is said to be set; when a pulse is applied to the "restore" input, the FF is said to be restored or cleared.

If the flip-flop is in the "set" state, the set output will be at the signal level and the reset output at the no signal level. The opposite is true for the "restore" condition. In some logical circuits, we shall be interested in only one of the outputs. In this case only the necessary output will be shown.

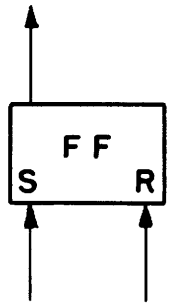


Fig. 6

Of course, the restore output line is still present in this case, but is not shown.

GATING AND BUFFING

Circuits known as gates are the chief form of switching used in the UNIVAC. As their name implies, gates permit or prohibit passage of signals from one point to another. Gates are indicated by the following symbol:

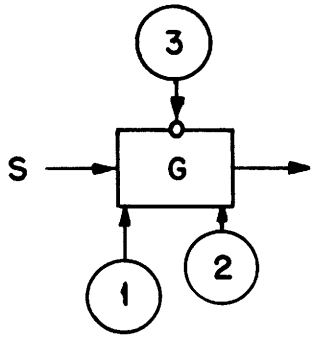


Fig. 7

The signal being gated (S) appears on the left. The various control signals are indicated as 1, 2, and 3. In order for signal S to pass through gate G, signals 1 and 2 must be present and signal 3 must be absent. Any other arrangements of signals is sufficient to prohibit signal S. Signal 3 is often called an inhibiting signal (note the small circle at the point of connection) while signals 1 and 2 are called permissive signals (without the circle connections).

The indications within the large circles always imply the existence of some other device which generates the required gate signals. Typically, another gate can generate such gate signals.

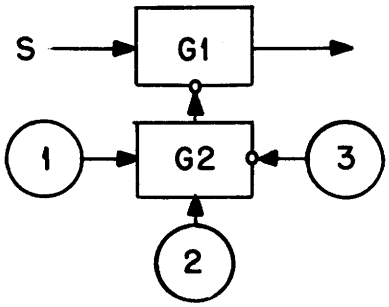


Fig. 8

Signal S can pass through G1 if G2 developes no signal. G2 can only develope a signal if 1 AND 2 are present AND signal 3 is absent. Gate circuits are sometimes called "and" circuits because they require the presence of this AND this AND this signal in order to operate. Gates may be "opened", "activated" or "excited" in order to permit signals to pass. They may also be "inhibited", or "closed", or they may "prohibit" signals from passing.

It is important to realize that every gate passes ONE signal under the influence of OTHER signals. In general, the control signals must overlap in time the signal being gated. The word "alerted" is applied to a gate to describe the condition when one or more but not all of its SEVERAL control signals are present.

A "pulse" signal is one which has a time duration of approximately one pulse time. Longer signals are generally referred to as static type signals.

For example, if a word is passing some point and the sign position is to be examined, then a gate with the desired pulse signal, say p_{37}, is connected to the point.

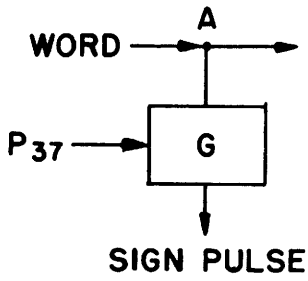


Fig. 9

If a pulse is present at point A during the sign time of a word, it is also gated through G by the p37 pulse. The p37 pulse for the gate is generated elsewhere in the computer.

The converse of gating is buffing. The buffer is indicated by a symbol B. A typical buffing circuit is shown as:

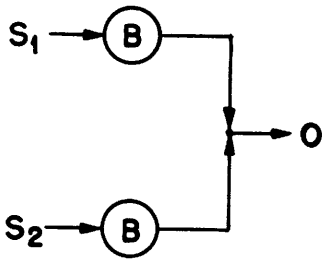


Fig. 10

The purpose of buffing is to combine several signal sources into a single line without interaction among the sources. Thus signal S1 cannot pass into S2 but only through its buffer B to the output O. Either signal S1 or signal S2 can pass into the output O. For this reason the buffing circuit is sometimes called an "or" circuit.

Gating signals can be applied through buffers, thus:

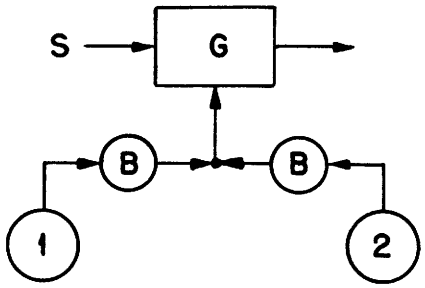


Fig. 11

Signal 1 or signal 2 may open or activate the gate to permit signal S to pass through the gate. At least one signal must be present; when both signals are present no new situation has been created.

It is not necessary to show buffing on the logical diagram if it is understood that such elements exist to prevent back-circuits. It will be assumed that no signal can be passed in the reversal direction of the arrows. Thus, figure 11 will be drawn as:

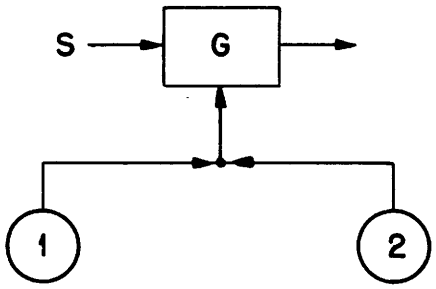


Fig. 12

To determine at some time after p37 if the flip-flop is set or reset and, hence, whether the word was positive or negative we must sample one of the output lines.

If we need the information at time p_1 to operate some other circuit we can sample a gate fed by the flip-flop with a p_1 pulse as in figure 13.

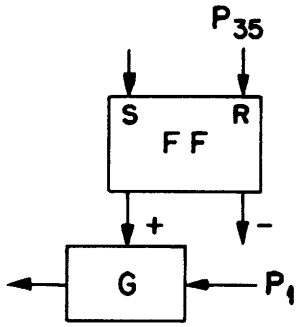


Fig. 13

A combination of gates and flip-flops can illustrate some basic circuits of a computer. A gating arrangement which is tested for the presence of a sign pulse was shown earlier. If the information obtained from the test (or the sampling as it is sometimes called) must be remembered then a circuit like this can be used:

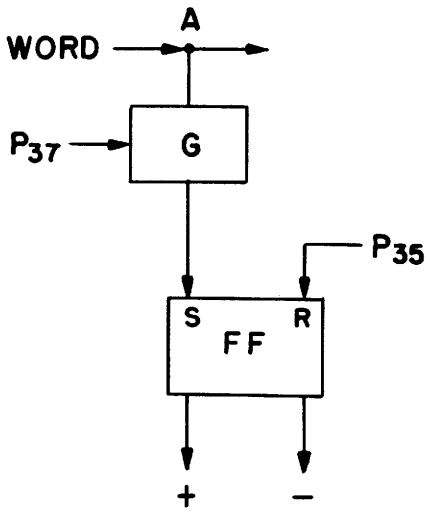


Fig. 14

If a sign pulse is present, it is gated by the p37 pulse to the S-input of the FF. The FF will remain set until the p35 pulse resets it. The p35 occurs two pulse times before the p37 during each word time. Therefore, the behavior of the FF is as follows: at all times the "-" or restored side of the FF is excited except when a sign pulse (plus sign) passes point A. When the sign pulse occurs the FF is set. (If there are 42 pulse times per word then the FF remains set for 40 pulse times each time a sign pulse passes point A.)

THE COMPARATOR (MAGNITUDE)

A binary magnitude comparator is a simple combination of gates and flip-flops. Its purpose is to compare the magnitude of two unsigned binary combinations. First, it is important to realize how to compare two binary quantities. Suppose the two quantities are:

A = 0 1 1 0 1 1 0 1 1 1

B = 1 1 1 0 1 0 1 0 0 1,

with the LSD's shown on the right. Begin by comparing the LSD's. They are both one's, therefore, the numbers so far are equal. Compare the second digits: A has a one, B has a zero; therefore, A is now larger. The third comparison is in favor of A and so forth, until the MSD is reached when the decision finally falls to B.

The circuit for this comparison is as follows:

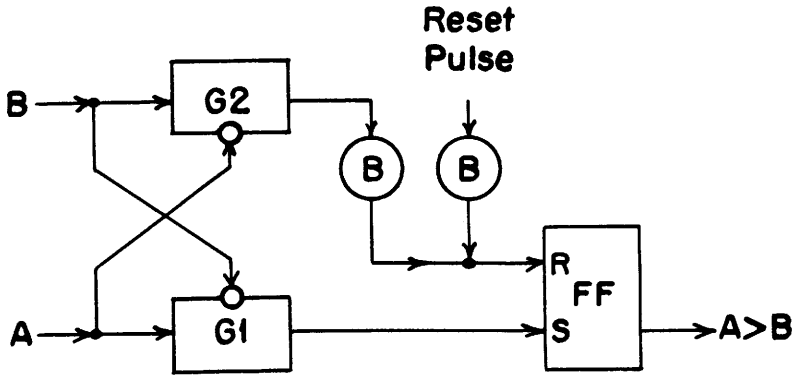


Fig. 15

B is assumed larger than A unless proved other wise; therefore, the restoring pulse removes the A > B before the comparison starts. As the first or LSD pulses of A and B reach their respective inputs, both gates, G1 and G2 are closed by the opposite input. No pulse reaches FF. On the second pair of digits the pulse on A inhibits G2 and passes through G1 to set FF. A now is greater than B. The third pair of digits follows the same procedure. When the fourth pair of digits arrive, the pulse in B inhibits G1 but passes through G2 to restore the FF, and so forth. When the MSD's are compared, the FF is left in restored condition and the fact is determined that A is <u>not</u> greater than B.

Suppose the two quantities A and B had been equal: no pulses would have reached FF and it would have remained restored. Therefore, the comparator only indicates two conclusions A>B and A≤B; it can not distinguish between A<B and A=B.

A more accomplished comparator is shown in Fig. 16.

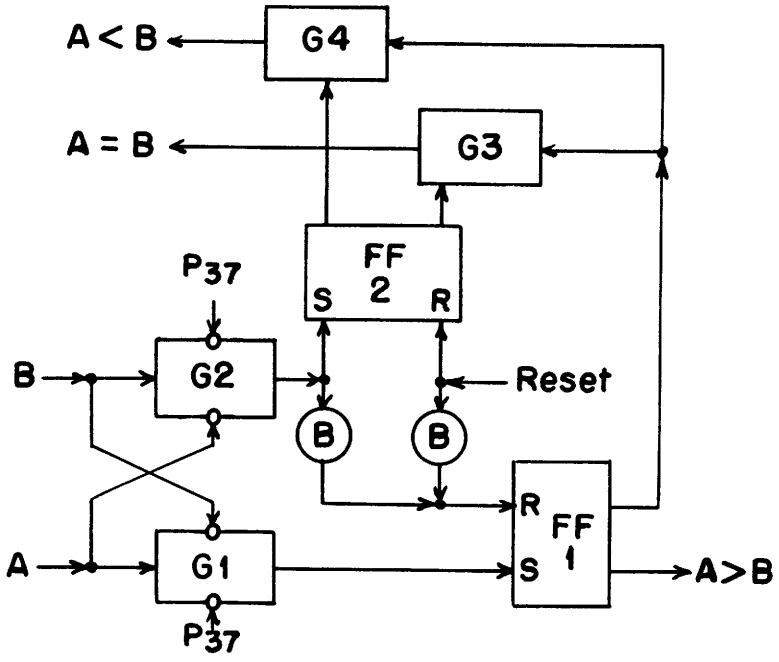


Fig. 16

When the restoring pulse restores FF1 and FF2, G3 is open and the A=B signal is given. This signal persists as long as identical zeros and ones appear at both A and B. If a one appears at input A with a zero at input B then FF1 is set. If the opposite event occurs then FF2 is set and FF1 is restored if it had been previously set and G4 is opened giving an A<B signal. Whatever state exists after the MSD's have been compared indicates the proper conclusion as to the magnitudes of the two quantities, A and B. This sign pulse suppression on G1 and G2 prevents a comparison of sign which can be compared by another type of circuit.

DELAY MECHANISM

If signals can exist in dynamic form, then an electrical delay constitutes a type of memory or storage facility. The common acoustic echo illustrates this storage phenomenon. For a period of time after generating a sound, the sound is stored in the form of acoustic waves which travel from the sound source toward a reflector. The reflector is unessential to the phenomenon but makes it easier to appreciate the storage principle. Suppose, instead of a reflector, there were some receiving device at a distance from the sound source. The same principle holds true. By the fact that the sound has experienced a delay, that is, requires a finite transit time for travel between source and receiver, it has been stored or remembered for a length of time equal to the transit time.

Ordinarily, the transit time for propagation of an electrical effect, through, say, a wire, is much too short to realize any practical or realizable storage capacity. However, there are several known methods by which the propagation of electrical effect can be retarded. One of these is to convert the electrical effects into acoustic patterns and thus make use of the much slower rate of acoustic propagation. The mercury memory with its transducing crystals at each end is typical of this type of storage device. The acoustic propagation through the mercury between the transmitting and receiving crystals is exactly analagous to the acoustic transmission described above.

Another type of delay is the electric delay line. Any coaxial cable with its distributed capacitance and inductance represents a delay line in which transit time for signals is longer than over ordinary wires. A convenient equivalent device requiring less physical space for a given delay time, is a lumped parameter delay line in which coils and capacitor are wired together as follows:

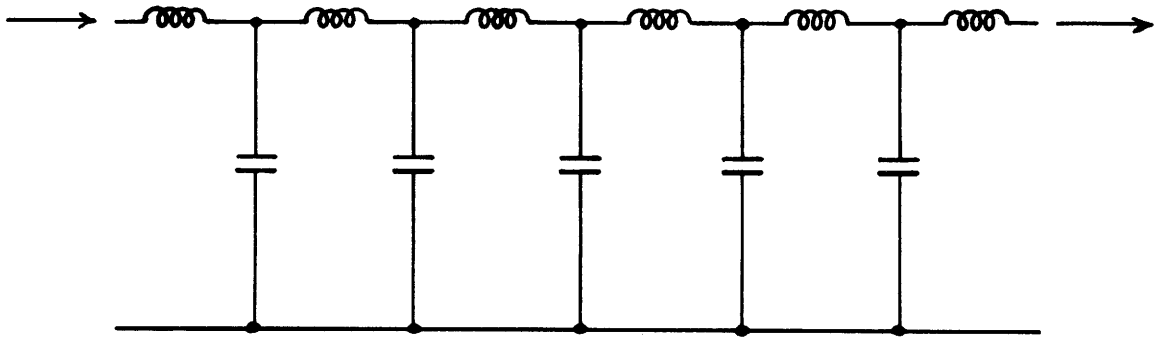


Fig. 17

The total delay of such a line is fixed by the number of sections. It is the pulse rate which determines how much storage such a line represents. The higher the pulse rate, the more pulses can be sent into the line before the first pulse reappears at the opposite end of the line.

The important concept of timing now becomes a matter of inserting and removing the correct amounts of delay in order to bring about the synchronous arrival of various pulses and signals at a given point.

In the logical diagrams, both acoustic and electromagnetic delays will be represented by boxes containing a number to indicate the delay time in pulse time units. Thus, figure 18 represents the delay of two pulse times.

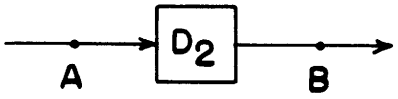


Fig. 18

If a pulse is fed into this line as a p_2, it will control the voltage of point A for time p_2, but will not control the voltage of point B until time p_4 or two pulse times after its introduction to point A.

ADDERS

By combining gates, buffers, and delays, several simple types of adders can be constructed. One example is called the half-adder. The half-adder combines the pulses of two words and produces either sum or carry pulses according to the laws of binary arithmetic. These rules are as follows:

0 + 0 = 0
0 + 1 = 1 + 0 = 1 (sum pulse)
1 + 1 = 0 (sum pulse) and 1 (carry pulse).

A half adder is shown in figure 19.

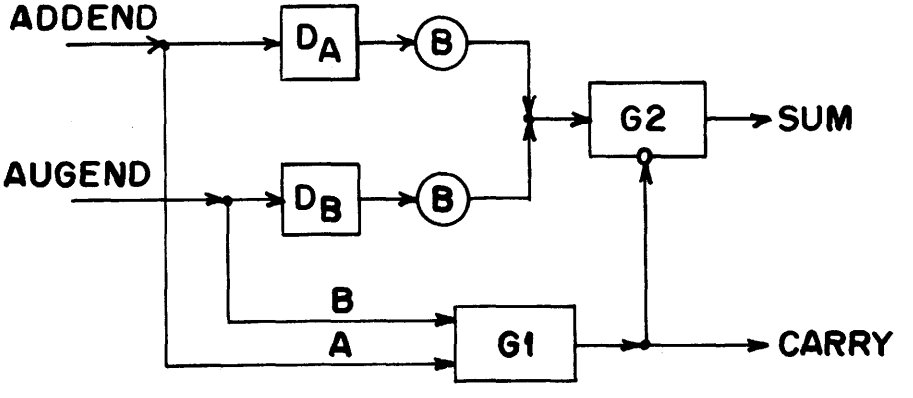


Fig. 19

The two inputs are marked addend and augend. If no pulses enter either input, no pulses are emitted. If a pulse appears on either input, it reaches G1 and also passes through the delay (D), a buffer to G2. Since G1 had only one input signal it did not open and, therefore, did not inhibit G2. Therefore, a sum pulse was produced.

If two pulses are applied to a half adder, one to each input simultaneously, then G1 does operate. The two pulses become one pulse in G1. The output of G1 becomes a carry pulse and also inhibits G2. The two input pulses pass through the delays and buffers to G2 but are prevented by G2 from passing to the sum output. The two delays are inserted in the path to G2 in order to delay the pulses for the amount of time required for G1 to develope its inhibition on G2 if both inputs of G1 have been excited. The half adder thus follows all the rules laid down for it above. The half adder will be shown on the logical diagrams by the following symbol:

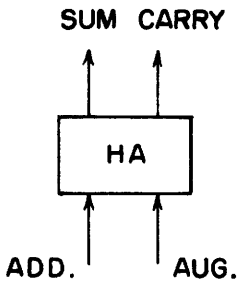


Fig. 20

In some cases the output of the carry line is not used and will be ommitted from the symbol. In a half adder, the missing operation is the transfer of the carry pulse back to the input of the adder so that it can be combined with the next digit pair entering the adder. First, such a device would require a one pulse delay between the carry output and the input of the adder. However, there is still another problem in that the half adder only has two inputs: the carry pulse could not be buffed into one of the two present inputs because the carry pulse would be lost if another pulse should occur as a digit of the word entering that word. Therefore, a third input must be devised to provide for the triple input conditions when there is a one at both the addend and the augend inputs and also a carry pulse to be added in from the previous addition. For this operation a full binary adder is required. The full binary adder can be constructed from two half adders, which is how the half adder received its name. An arrangement of two half adders which forms a full binary adder is shown in figure 21.

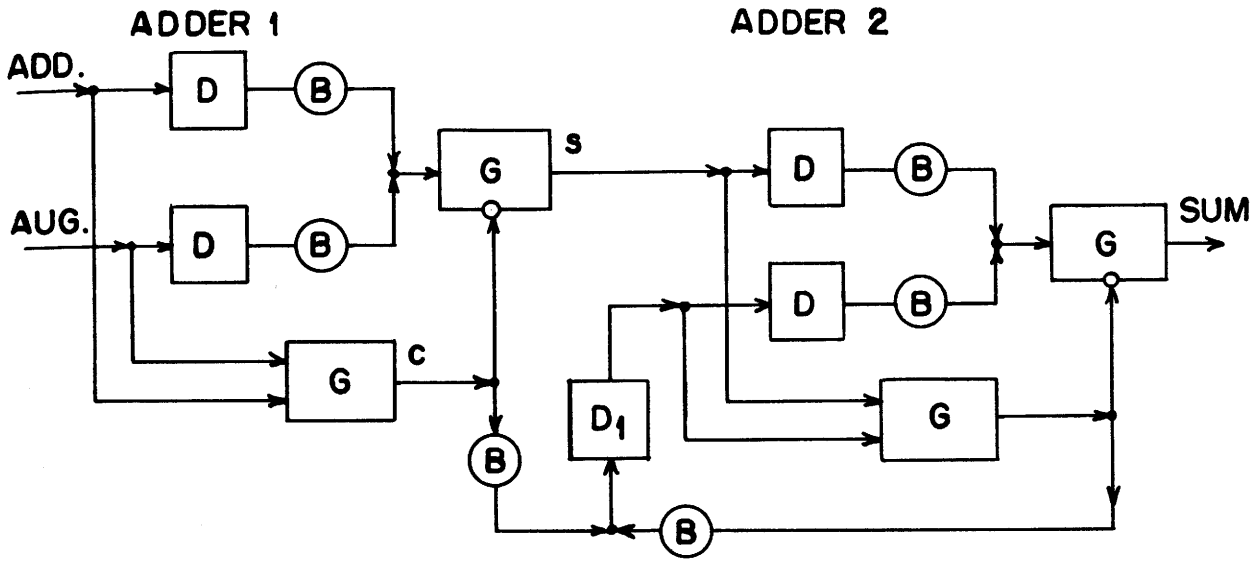


Fig. 21

Adder 1 combines the input digits to form the proper sum. If a sum pulse occurs it passes directly into and through adder 2 to become a final sum pulse. If a carry pulse occurs from adder 1, it enters a one pulse delay, D1, for storage until the next pair of digits are added in adder 1. If they produce a sum pulse from adder 1, then the sum pulse and the carry pulse formed from the previous addition operation are combined in adder 2. This situation, incidentally would produce a carry pulse from adder 2 and no final sum pulse. If the carry pulse occurs when zeros enter the addend and augend input at the next digit time, then the carry pulse passes through adder 2 to become a final sum pulse.

On the logical diagrams a full binary adder will be represented by the following symbol:

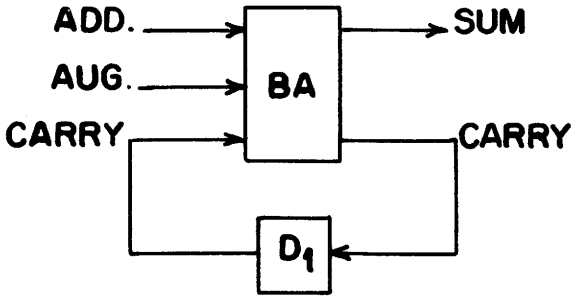


Fig. 22

COMPLEMENTING

The half adder can be used for another purpose in binary computing. The rule of the half adder is that two "ones" produce a zero on the sum output, and that a "one" and "zero" in either order on the inputs produce a one. Now examine the binary complement. By definition, the complement, when added to the original quantity should produce all zeros except a final carryover in the place beyond the most significant digit (MSD). Thus, consider the following:

```
quantity   = 1 1 0 1 1 0 0 0 1 1 1 0 1
complement = 0 0 1 0 0 1 1 1 0 0 0 1 1
             1 0 0 0 0 0 0 0 0 0 0 0 0
```

The upper quantity is represented in complementary form by the lower quantity. Note that the complement is formed by interchanging ones for zeros and zeros for ones in the original quantity except for the least significant digit (LSD). The particular complement given in the illustration is called a binary complement. The complement by interchanging ones and zeros without correction in the LSD position is the binary complement minus one. Thus:

```
original quantity = 1 1 0 1 1 0 0 0 1 1 1 0 1

binary complement minus one = 0 0 1 0 0 1 1 1 0 0 0 1 0
                              1 1 1 1 1 1 1 1 1 1 1 1 1
```

The binary complement is used more frequently than the binary complement minus one. The binary complement minus one can always be converted to a binary complement by adding a one to the LSD.

Consider now the complements and the half adder. If a binary quantity is added to a string of ones the binary complement minus one is obtained since the half adder has no carryover circuit. Thus, only the individual sums are obtained for each pair of corresponding digits.

```
                                  1 1 0 1 1 0 0 0 1 1 1 0 1

                                  1 1 1 1 1 1 1 1 1 1 1 1 1
binary complement minus one   = 0 0 1 0 0 1 1 1 0 0 0 1 0
complement correction         =                         1
binary complement             = 0 0 1 0 0 1 1 1 0 0 0 1 1
```

By means of a half adder the binary complement minus one can be formed by "adding" the original quantity to a string of ones. The binary complement can be obtained by adding a one to the LSD. Thus the half adder is also a complementer. When so used, the carry output of the complementer is ignored.

Consider now the following situation. If two pulse trains are identical, and are applied simultaneously to the two inputs, the sum output of a half adder should emit no pulses. The combinations which produce no sum pulses are two zeros or two ones. Thus, the half adder can be used as a comparator. If the two input quantities are identical, no sum pulses occur. If either quantity contains a one where the other has a zero, the sum output emits a pulse which signals the lack of identity.

The half adder therefore, performs two very useful functions for a serial binary computer: it complements and it compares for identity.

COUNTING

One of the devices required in a computer is a counter which can accept a sequence of pulses and indicate at any time how many pulses have been received. Since the computer operates in the binary system, the binary counter is the basic counting device. The binary counter (BC) is indicated on diagrams by either of the following symbols:

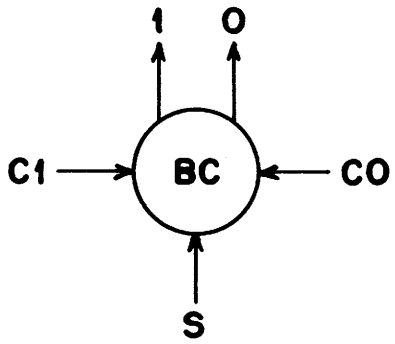


Fig. 23

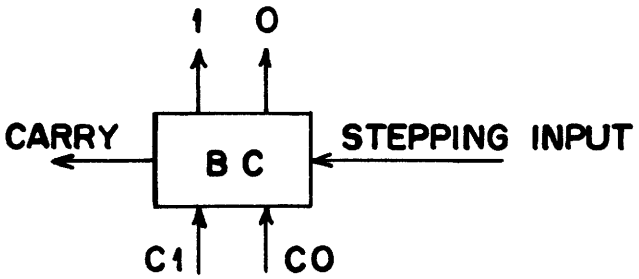


Fig. 24

There may be as many as three inputs to a binary counter. It resembles a flip-flop in that it has two stable conditions. However, it differs from the flip-flop in its basic logical significance. A binary counter always changes state when it is pulsed on the main input(s). The flip-flop is put into either state without specific knowledge of its previous state, while the binary counter cannot be so controlled by its main input.

Each binary counter can be cleared to zero, or to one, if desired, by special inputs, CO and C1. A single binary counter may be considered as an oddness-eveness indicator. For example, if the binary counter (BC) has been cleared to zero initially, then, any time after an even number of pulses has been applied to the input, the BC will read 0; conversely, an odd number of pulses will find the BC reading one.

More easily recognized binary counting occurs when a group of binary counters are connected as a multi-state counter as in figure 25.

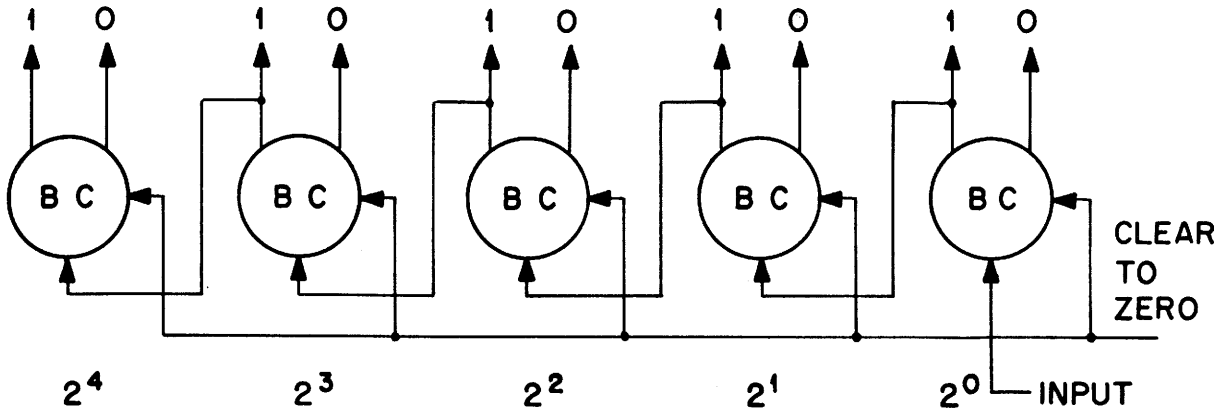


Fig. 25

Such an array has 32 different stable states as follows:

0	0	0	0	0	=	0
0	0	0	0	1	=	1
0	0	0	1	0	=	2
0	0	0	1	1	=	3
0	0	1	0	0	=	4
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
1	1	0	0	1	=	25
1	1	0	1	0	=	26
1	1	0	1	1	=	27
1	1	1	0	0	=	28
1	1	1	0	1	=	29
1	1	1	1	0	=	30
1	1	1	1	1	=	31

Each time any one counter of the group is pulsed and it passes from one to zero, the one output delivers a pulse to the input of the next higher stage of the counter. When all the stages have reached one (which in the case of five counters equals decimal 31) all stages produce carry pulses which turn every stage back to zero. At any time after a string of pulses has been applied to the LSD counter input, the 5-stage counter illustrated, will indicate how many pulses up to 31 have been applied. Everytime a multiple of 32 pulses occurs the counter clears to zero and starts counting again.

By increasing the number of stages such a binary counter could count up to any power of two where the number of stages would equal the power. Thus, a 5-stage counter has 2^5 or 32 stable states; an 8-stage counter would have 2^8 or 256 stable states.

The method by which such a counter is read or interpreted depends upon how the information is to be used. Various uses for binary counters will be given later and the different methods for reading them will be discussed when appropriate.

REGISTERS

The most elementary static binary memory has already been described as the flip-flop. For remembering an entire word, another type of binary memory can be employed. In general, such a device is called a register. The register is simply a combination of gates and delays hooked in a loop such that any pulse entering the loop or register experiences exactly one word time of delay in one complete circulation.

One additional element is introduced at this point called a pulse former (PFR). The pulse former is what adds the energy of circulation to the system. Passage of pulses through any of the elements, such as gates or delays, causes deterioration of the pulses. The proper shape and timing of pulses is dependent upon the PFR's which are introduced at many points throughout the computer. They serve no logical purpose, in general, but do lift the block diagrams out of the realm of purely imaginary operation into the realm of practicality.

The pulse formers are controlled by timing pulses from the master oscillator. In the small computer we are considering these occur at the rate of one million per second, or at intervals of one microsecond. These timing pulses are sufficient to insure that information pulses reshaped by the pulse formers will be exactly of one microsecond duration.

A pulse former will be shown on the diagram by the following symbols:

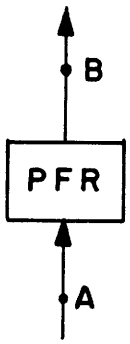


Fig. 26

A pulse former will delay the transmission of pulses for one pulse time. Timewise, it has the effect of a one pulse delay. If in figure 26 a pulse is fed into the pulse former during time p_2, it will not control the voltage at point B until time p_3.

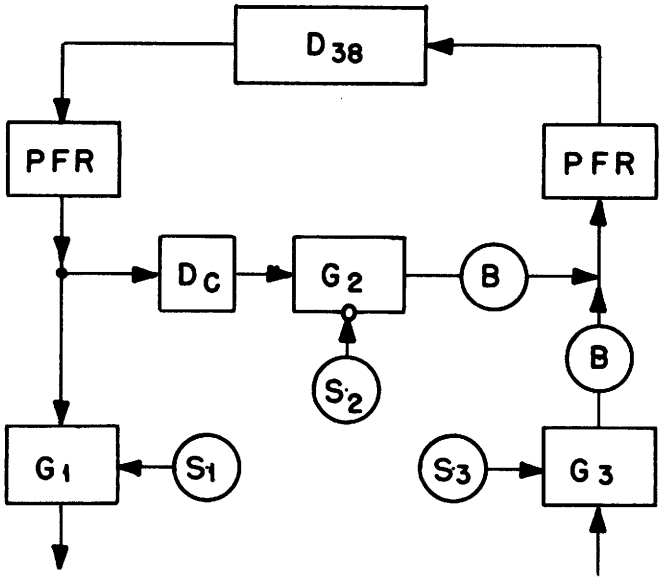


Fig. 27

A typical register for a binary computer is shown in figure 27. The direction of circulation in the register is determined by the polarization of the PFR's and presence of buffers (B). If the word length is 42 pulses then the register must represent exactly 42 pulse times delay. The main delay element (which could be an electric delay line or a mercury acoustic line) is 38 pulse times long. Each pulse former represents 1 pulse time of delay and the compensating delay (Dc) represents 2 pulse delay or a total of 42 pulse times for a complete circulation.

There are three gates, G1, G2, and G3. G1 is the output gate and G3 is the input gate. G2 is called the clear gate. Its function is to interrupt the path of circulation so that any pulses in the register are cleared out. The clear gate is controlled by signal S2; when S2 is present G2 is inhibited and the register is cleared. The duration of S2 is important, for the signal S2 must certainly last at least as long as the time required for a word to pass the clear gate.

The output gate, G1, is controlled by S1. When S1 is present, the pulses passing the output gate connection can be communicated to another register or other elements of the computer. The operation of G1 has no effect on the pulses in the register.

The input gate, G3, is controlled by S3. Usually S2 and S3 occur together because each transfer to a register such as here illustrated, requires that the previous contents of the register be cleared out before the new word enters. Actually the duration and existence of signal S2 and S3 can be identical.

Since G2 and G3 are at the same time point within the register (there being no significant delay between these two gates) the input can be opened at the same time that the clear gate is closed or inhibited. If G2 and G3 are at the same point in time, and the direction of circulation within the register is as shown, then there must be exactly one word delay between G3 and G2 from the standpoint of an entering word. The existence of the buffer prevents the entering or incoming word from reaching G2 except by the long route.

Now, if the two gates are operated together, clearing of the first pulse position of the previous word begins just as the first pulse of new word passes through input gate G3. Furthermore, G3 should close just after the last pulse of the incoming word. But at the same time, the last pulse of the old word in the register has just reached the clear gate.

Since the first and last pulses of the information portion a word in a register are separated by the space between words (SBW), there are several pulse times available during which to close the input gate and open the clear gate. The clear gate, G2, must certainly open before the first pulse of the new word reaches it. Due to the precise timing throughout the computer, the time relation between the old word in the register and the new word entering a register is identical.

The timewise separation of G1 and G2 by the D2 delay will be explained later.

SHIFTING

In the ordinary desk calculator, the operation known as shifting is accomplished by moving the accumulator dials on the carriage with respect to the input keyboard. The shifting operation is required in multiplication and division. Effectively, shifting is simply multiplying a quantity by a power of the base of the number system. Thus a decimal calculator multiplies by powers of 10 (positive and negative) as the carriage is moved to the left or right.

In a binary device a shift of one digit position to the right is effectively multiplying the quantity by 1/2 or 2^{-1}. Shifting left two places is multiplication by 4 or 2^2. The method of shifting in a dynamic circulating register is accomplished by adding or removing unit pulse times of delay. Thus in figure 28, shifting circuits have been added to a register. The normal path of circulation is through gate G5.

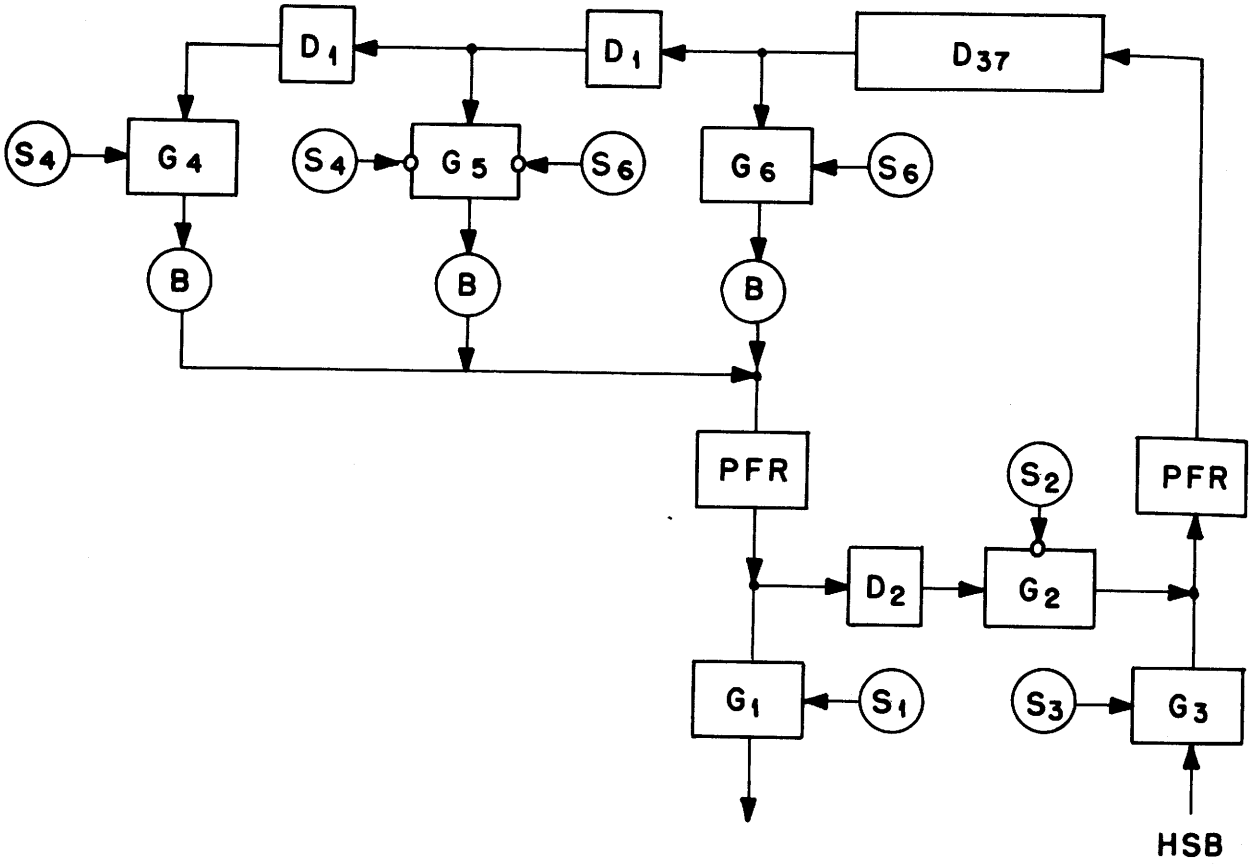


Fig. 28

DISTRIBUTION AND COLLECTION

The flip-flop and binary counter were shown to be a form of static binary memory. The circulating register is a form of dynamic memory. If both types of devices exist within the computer, there must be some means of converting from dynamic to static storage and from static to dynamic storage. The former process is called distribution and the latter is called collection.

Figure 29 shows the schematic arrangement for distribution. First, there is an electrical delay line shown on the left. There is a delay of one pulse time between each tap. The connection from the delay line to the gates are called taps.

When a sequence of pulses enters the delay line, the signal S can be so timed that each gate (all gates are sampled at the same instant of time by S) will be operated by the presence of a pulse at its tap. If a pulse exists at any tap, it is transferred through the gate to the elementary binary memory (M). These memories (M), for example, can be flip-flops. They might be binary counters, in which case the special clearing inputs of the BC's would be used and not the stepping inputs.

Suppose we wish to set up the p_7, p_8, p_9, and p_{10} positions of the computer word in M_4, M_3, M_2 and M_1. P_7 controls the voltage at point A during p_7, but does not control the voltage at point B until time p_8. During p_8 time, p_8 controls the voltage at point A. Two pulse times later at time p_{10}, the p_7 pulse will control the voltage of point D, p_8 the voltage of point C, p_9 the voltage of point B and p_{10} the voltage of point A. Hence, if we sample the gates at p_{10} ($S = p_{10}$), pulses present in any or all of these four positions will be transferred into the desired memory, $M_1 \ldots M_4$. Although the sequence of pulses continues to move through the delay line, no other pulse positions of the computer word can be transferred to the binary memories, since the gates are sampled only during time p_{10}.

Therefore, a word circulating in a register can be converted into static storage by using a distribution line, a gating system and a set of flip-flops or other binary memories. The time length of the delay line must equal the numbers of pulses being converted; the number of gates and binary memories must likewise equal the number of pulses in the sequence.

The process of collection is the opposite of distribution. Figure 30 shows the schematic arrangement for collection. Information is stored in the binary memories M_1, M_2, M_3 and M_4. One output of each memory element is fed to a gate. Signal S samples each gate during successive pulse times, producing an output pulse on the transmission line, for each gate alerted by its memory element.

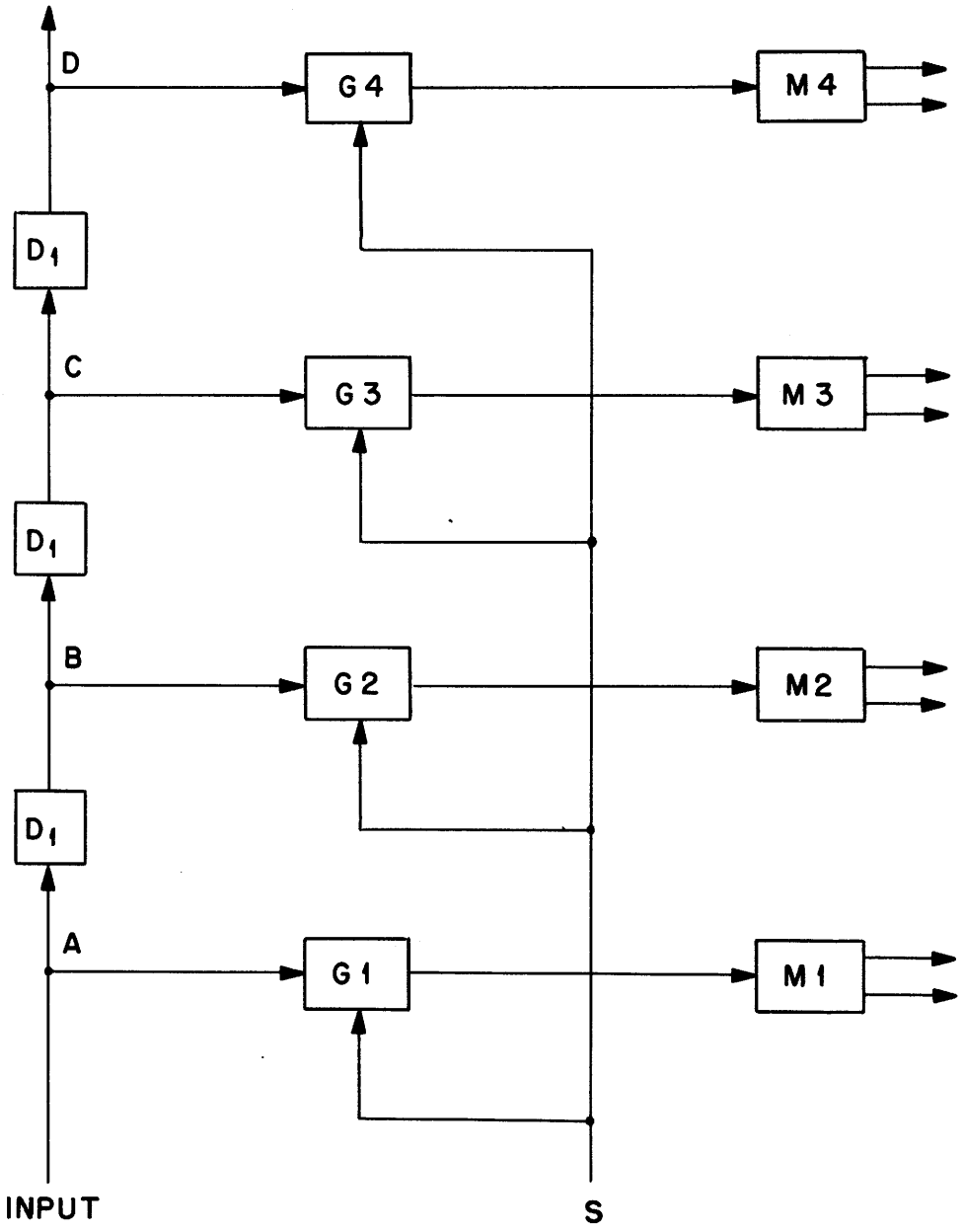


FIG. 29

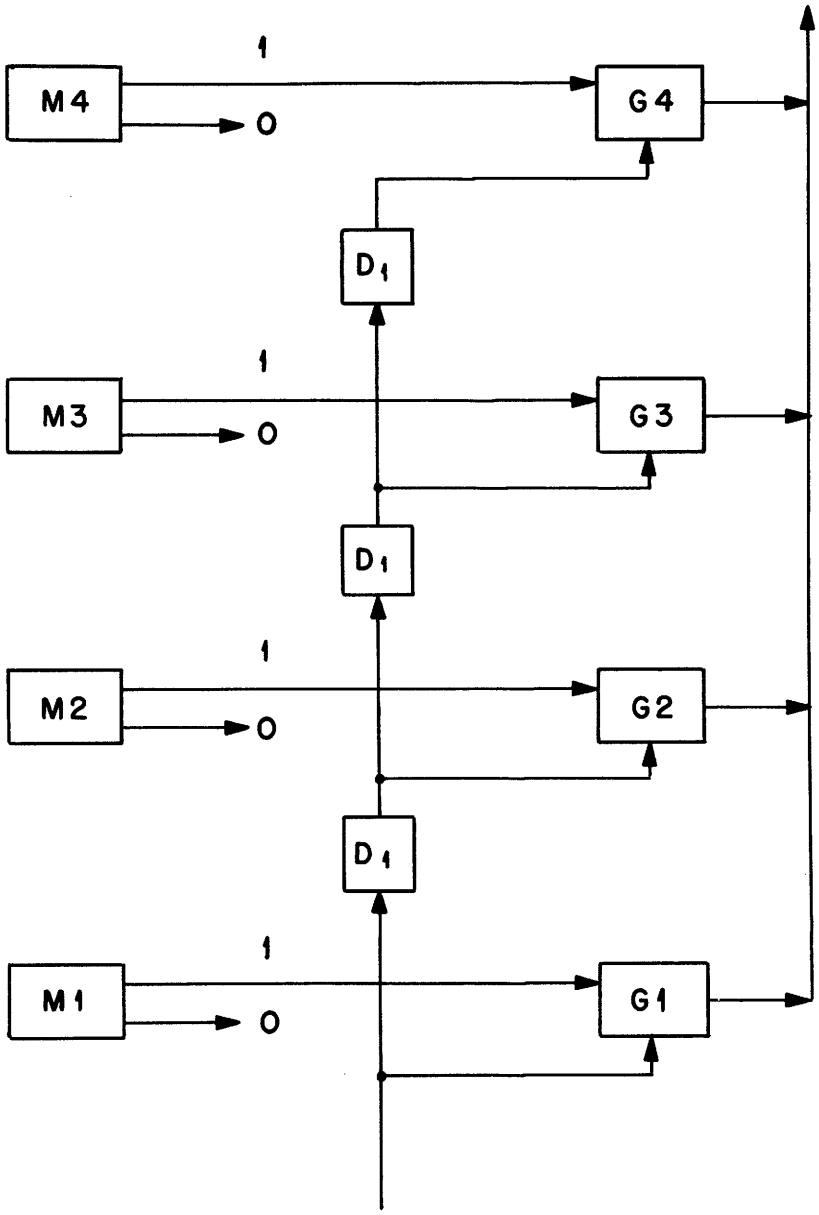


FIG. 30

If the p_{10} position of a word is stored in M_4, p_9 in M_3, p_8 in M_2 and p_7 in M_1, we can convert this to a sequence of pulses by making $S = p_7$. Then G1 is sampled at p_7 and will produce a pulse if the state of M_1 indicates a binary one. G2 is sampled at p_8, G3 at p_9 and G4 at p_{10}.

Only one output of each memory element is used to alert its gate. Hence, a gate will have an output only if the "1" output line of its memory element is excited, even though all gates are sampled by signal S.

FUNCTION TABLES

A function table (FT) is a device which can decode many input lines into a single output line or encode a single input line into many output lines. The former type function table is called a decoding FT while the latter is called an encoding FT.

Consider the output lines of two flip-flops. The excited not-excited combination of these lines enables us to represent numbers 0, 1, 2, 3, as shown by the following table.

FF-2	FF-1	Numbers
R	R	00
R	S	01
S	R	10
S	S	11

The decoding of such an input into single output lines is shown schematically in figure 31.

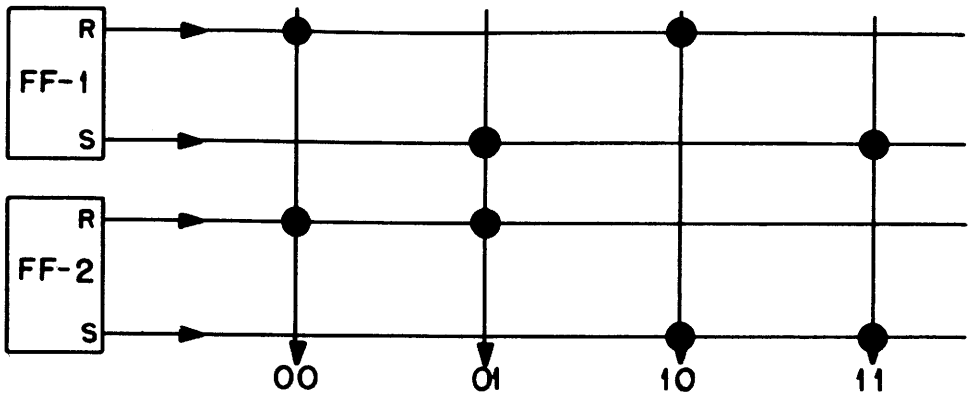


Fig. 31

Only one output line of each flip-flop can be excited at any instant of time. For a given output line to be excited, both input lines to which it is connected (dots indicate connections) must be excited. The connections may be either resistors or diodes.

If a group of flip-flops are set up, for example, by a distribution circuit, then the quantity stored in the flip-flops can be read out on single lines by means of a decoding function table. Similarly, a multi-stage binary counter can be "read out" or decoded by the same type function table.

The encoding function table uses a single excited input line to "pick up" or excite all the output lines connected to it. Figure 32 shows an encoding table driven by the four output lines of the decoding table shown in figure 31.

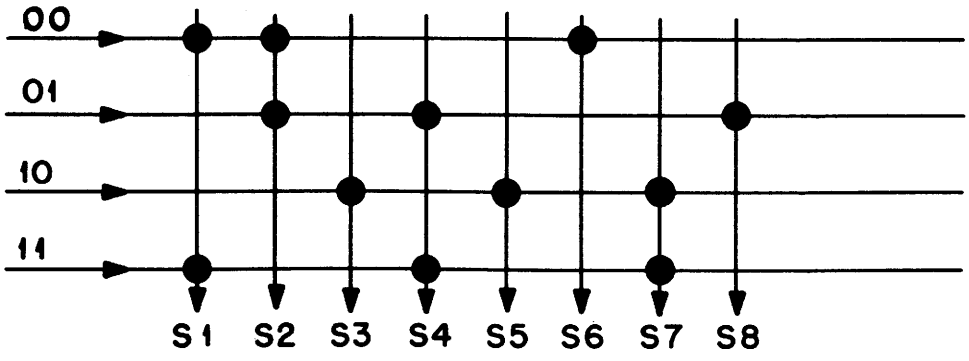


Fig. 32

Because of the nature of the decoding function table, which provides the input in figure 32, only one input line can be excited at any instant of time. Thus, if the "00" line is excited, the S_1, S_2 and S_6 lines will be picked up. The "10" line will pick up S_3, S_5 and S_7, but no others. This makes it possible for a single input line to excite a pattern of output signals which will cause the computer to carry out some pre-determined operation.

Sec. 6. Operation of the Computer

The Instruction Code

The computer was designed to carry out only eight instructions, in order to keep the logical circuits as simple as possible. Therefore, only the p_{30} to p_{36} positions of a word are decoded as an instruction. The p_{30} to p_{33} positions indicate the address of a memory location if the memory is to be involved in the instruction. Otherwise, these positions are not used by the computer. When a word from the memory is needed, the p_{32}, p_{33} position indicate which of the four memory channels contains the word, and the p_{30}, p_{31} position indicate which of the four within the selected channel is to be read out. Thus, an address of 1000 would indicate the zero word of channel two. Or if we number the words from 0 (for the zero word of the zero channel) through 15 (for the 3 word of channel 3), 1000 would indicate word 8 of the memory.

The eight possible pulse combinations of the p_{30} to p_{36} position gives the code for the eight instructions the computer will execute.

Figure 33 indicates the portion of a word which the computer can use in instruction.

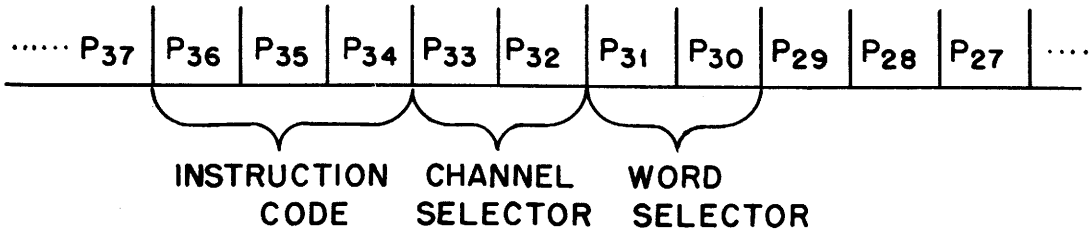


Fig. 33

The code is described as follows:

Instruction	Description
000m	Transfer the contents of the memory location indicated m to rA, clearing rA of its former contents and leaving (m) unaltered.
001m	Transfer the contents of rA to the memory location indicated, clearing m of its former contents and leaving rA unaltered.
010 0000	Transfer the contents of rA to rB, clearing rB of its former contents and leaving rA unaltered.
011m	Transfer the address m of the instruction word from the static register to CC, clearing CC of its former contents.
100m	Transfer the contents of memory location m to rB clearing rB of its former contents; transfer (rB) and (rA) to the adder and return the sum to rA, clearing rA of its former contents and leaving rB unchanged.
101m	Transfer (rA) and (rB) to the comparator leaving the contents of both registers unchanged; if (rA) > (rB) transfer the address m of the instruction word from the static register to CC, clearing CC of its former contents if rA < rB do not change CC.
110 0000	Shift (rA) one digit position to the right replacing the sign position with binary zero.
111 0000	Shift (rA) one digit position to the left replacing the LSD with binary zero.

Organization of the Computer - The overall layout of the computer is shown in figure 34.

Memory - The main memory consists of four acoustic delay lines each capable of storing four words, or a total storage of sixteen words. These words can be instructions or data.

All instructions are stored in the memory as numbers, and are distinguished from data only by the manner in which the computer makes use of them. There is no section of the memory specifically allocated for instructions; data or instructions can be stored in any memory location.

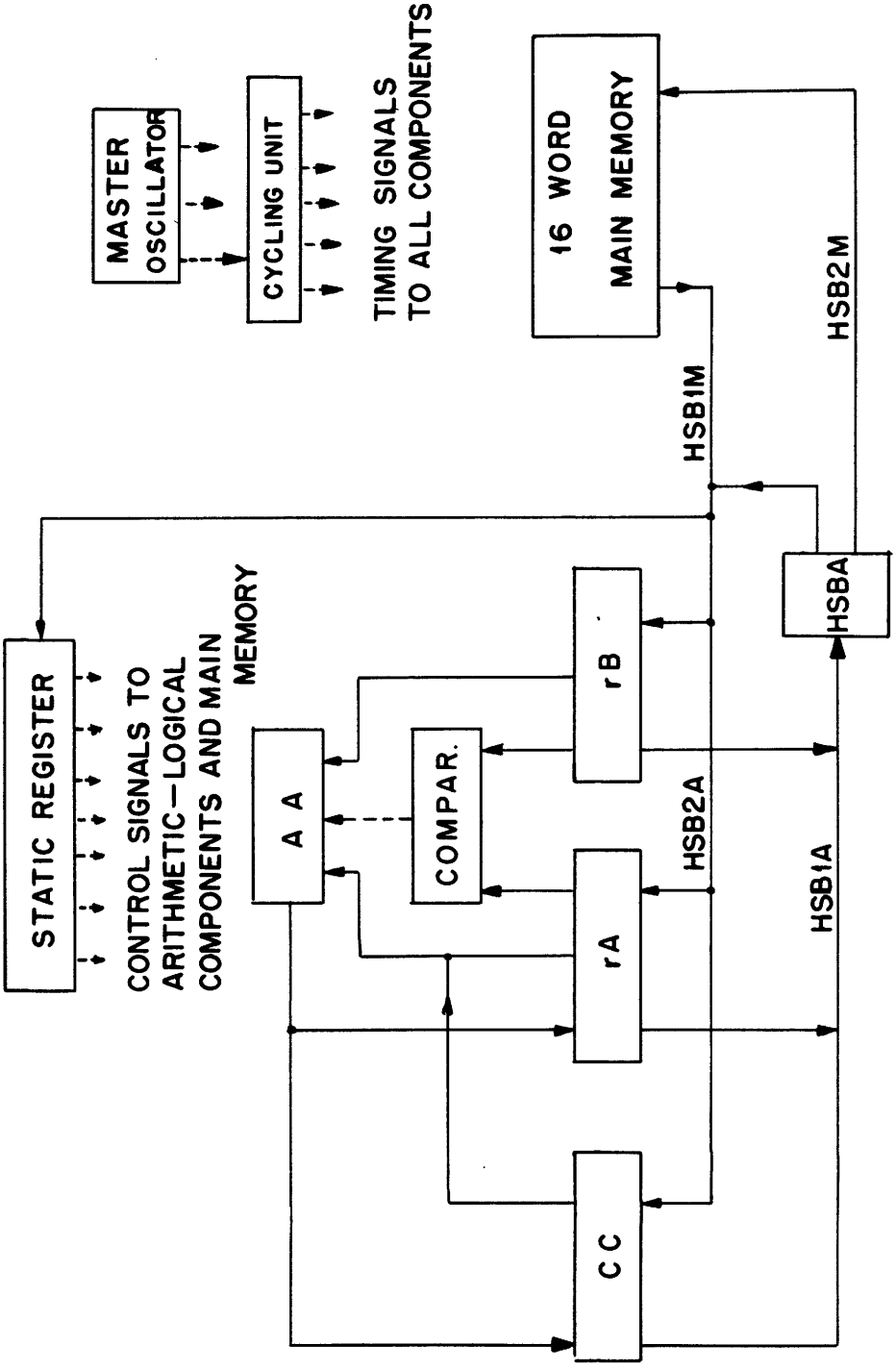
It should be clearly understood that no arithmetic or logical operations are carried out in the memory. It is used only for storage. When a word stored in the memory is required as an operand or for decoding as an instruction, it must be transferred to the arithmetic circuits or control circuits.

The transmission line which carries information from the memory is called high speed bus (HSB) 1M. The line which transmits information to the memory is HSB2M.

Arithmetic Circuits - These circuits consist of the algebraic adder (AA), the comparator (CP), and two one-word acoustic type storages called register A, rA, and register B, rB. The latter serve as temporary storages for words to be used by the adder or comparator.

The comparator is used on the 101m order to compare the absolute values of rA and rB, and initiates one of two possible sequences of instructions based on (rA) > (rB) or (rA) < (rB). The comparator is also used on the 100m order to determine whether addition or subtraction is to be performed by AA.

The adder is used on the 100m order to add or subtract the quantities stored in rA and rV, returning the sum to rA. It is also used by the control circuits to increase the word stored in the control counter (CC).



BLOCK DIAGRAM OF A SIMPLE BINARY COMPUTER

FIG. 34

Control Circuits - The control circuits consist of the static register (SR), the cycling unit (CU), and the control counter (CC). The static register converts the $p_{30}...p_{36}$ positions of a word to be used as an instruction into flip-flop storage. These flip-flops drive function tables to produce the necessary signals to carry out the instruction.

The cycling unit generates timing signals which are sent to all components of the computer to synchronize its operations.

The control counter is a one-word, mercury delay line register. Its purpose is to store the address of the next instruction word. The numerical value stored in CC is referred to as the CC-reading. Normally, the instructions are performed according to the numerical value of the memory locations in which they are stored. Thus, if CC initially reads zero, the computer is referred to 0000 for the first instruction word. As the reference to 0000 is completed, the CC-reading is advanced to 0001 and so forth. Hence, CC functions as the sequencing mechanism of the computer.

If the normal sequence of instructions is to be changed, then the CC-reading must be altered. Both the 011m and 101m orders accomplish this. (See Instruction Code)

The transmission line which carries information to rA, rB, CC and SR is HSB2A. HSB1A receives information from rA, rB or CC and transmits it to the high speed bus amplifier (HSBA). This is primarily a switching central and can send information from HSB2A to HSB1A (for a register to register transfer) or to HSB2M (for a register to memory transfer).

Timing - Cycling Unit Signals

We have assumed a pulse rate of 1 Mg. for the computer. Pulses are produced at this frequency by a crystal controlled master oscillator, which maintains the pulse frequency with negligible variance. The pulse output of the master oscillator (one per μsec.) is not used to represent information directly, but controls all pulse formers in the computer to limit their output time for one information pulse to exactly one microsecond.

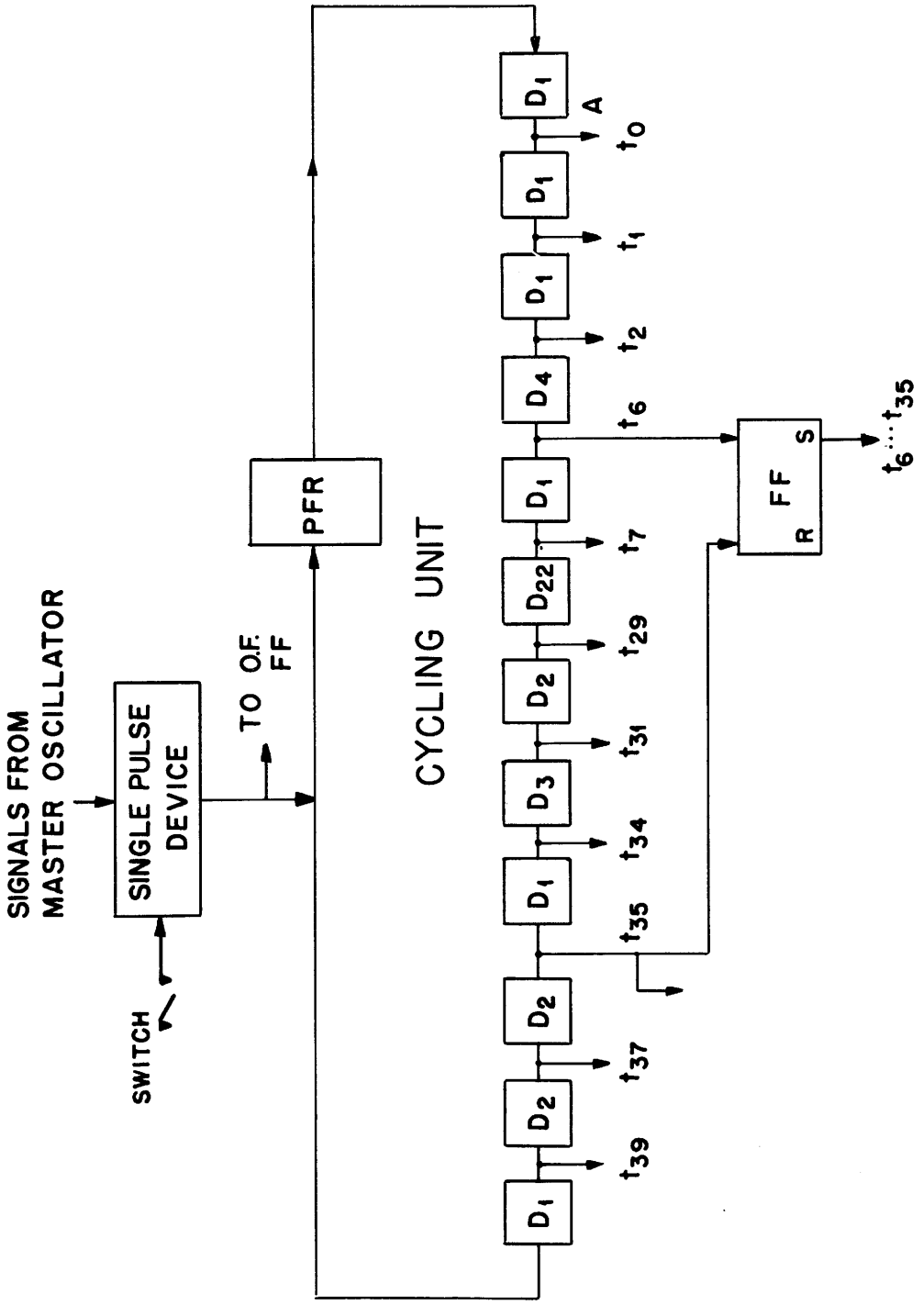

SIGNALS FROM MASTER OSCILLATOR
SINGLE PULSE DEVICE
SWITCH
TO O.F. FF
PFR
CYCLING UNIT
D1
D1
D1
D4
D1
D22
D2
D3
D1
D2
D2
D1
A
t0
t1
t2
t6
t7
t29
t31
t34
t35
t37
t39
FF
S
R
t6 ··· t35

FIG. 35

The propagation of information pulses through the various components of the computer will tend to distort them, but pulse formers are inserted sufficiently often in the circuits to reshape them and keep their time duration constant.

For control purposes, it is necessary to have signals which occur at less than the basic pulse frequency of the computer. It is necessary, for example, to have a signal to mark the beginning of each minor cycle (word time) to indicate when p_0 position of a word in any of the one-word registers is available for read-out. Such signals will be produced by the cycling unit shown in figure 35. This is nothing more than a one-word register with taps needed for each signal.

After the master oscillator is started, the single pulse device is switch operated. This produces one pulse whose duration is fixed at one microsecond by successive pulses from the master oscillator. The single pulse is fed into the recirculation loop of the cycling unit. This passes a PFR and a D1 delay, and then produces a signal at tap A for one pulse time. The signal on this line is called a t_0 and indicates the beginning of a minor cycle. After 42 μsec. or one minor cycle, the pulse once more reaches tap A and the t_0 signal is repeated.

Additional signals are needed at various times within the minor cycle and these are produced by tapping off the recirculation loop an appropriate amount of delay after the t_0.

For timing signals which last more than a pulse time, but which repeat each minor cycle, flip-flops controlled by CU signals can be used. Thus, a signal is needed which lasts from t_6 through t_{35} of each minor cycle. This is produced by using a t_6 to a set of flip-flops and a t_{35} to reset it. Thus, the signal condition of the set output of the flip-flop is present only during the required time in each minor cycle.

Memory Timing

The main memory of the computer consists of four four-word registers or channels. The channels are numbered 00, 01, 10, and 11. Channel 00 is shown in full on Chart A while only the input and output lines of the other channels are indicated.

Each channel consists of a recirculation loop, a read-in gate, G_I; a read-out gate, G_0; a clear gate G_C; a control gate G_T. The recirculation loop contains 4 x 42 μsec. = 168 μsec. of delay. Thus, a particular word in a channel is available at the read-out gate once in each four minor cycles. The time for the transit of a word from the read-out gate, through the recirculation loop and both to the read-out gate (four minor cycles) is called a major cycle.

It is beyond the scope of this description to deal with the problem of getting information into and out of the computer. It will be assumed that all memory channels have been filled with information in such a manner that the p_0 position of a word is available at the read-out gate, G_0, at time t_0 is indicated by the cycling unit. This "pulse position-time" reference will be indicated on drawings by indicating at the appropriate point in the circuit (in this case, G_0,) that $p_0 = t_0$. The means that the p_0 position of any word in the recirculation loop will arrive at G_0 at t_0 of some minor cycle. It also indicates that the p_1 position is available at t_1, p_{36} at t_{36}, etc.

This reference timing does not necessarily mean that the p_0 pulse will always be the first read-out. If G_0 did not become permissive until t_3, the reference timing would indicate that the p_3 pulse is the first to pass through the gate.

It is now possible to explain the D2 delay in the recirculation path of the one-word register, which separates their read-in and read-out gates. The read-out time from the memory to the transmission line HSB1M has been fixed at $p_0 = t_0$. It is necessary to have a PFR in this transmission line, hence, the time on arrival at HSB2A which feeds the input gates of all one-word registers is $p_0 = t_0$. Thus, if

a p_0 pulse passes a memory read-out gate at time t_0 as indicated by the cycling unit, it does not arrive at the input gate of a register until time t_1 as indicated by the cycling unit due to the one pulse delay inherent in the PFR.

There is no delay, however, between the G_0, and G_1 in the memory channels, hence, information passing from a one-word register to a memory channel must arrive with a reference timing identical with the read-out timing, $p_0 = t_0$. But is necessary to have a PFR in the transmission line, HSB2A, which received information from the one-word registers. In order for a p_0 pulse to arrive on HSB2M until the proper timing $p_0 = t_0$ it is necessary to start it from the register a pulse time early in order to compensate for the delay of the PFR or at time $p_0 = t_{41}$. The D2 delay thus separates the read-in gate and read-out gates of the one-word register by the necessary two pulse times.

Switching Time

Most of the gates in the computer are controlled by signals from the encoding function table. The time required for the FT output signals to reach their new levels following any charge in the FT input is called switching time. The switching time allowed for the encoding FT signals is one minor cycle even though they reach equilibrium in less time. The control signals on certain gates must only change during the time the SBW is applied to the gates. A special flip-flop controls all such gates. This flip-flop is called the time-out flip-flop (FF-TO) and is always set whenever the input to the encoding FT is changed. FFTO is always set by a t_0 signal from the cycling unit and reset by the following t_0, hence, it established the one minor cycle switching time.

The set output of FFTO is inhibitory on most gates, so although FT signals may alert a gate during the time-out minor cycle, the gate is not able to pass information until FFTO is reset by the t_0 cycling unit pulse. This ensures that information will always leave a register LSD first.

The Three Stage Cycle of Operation

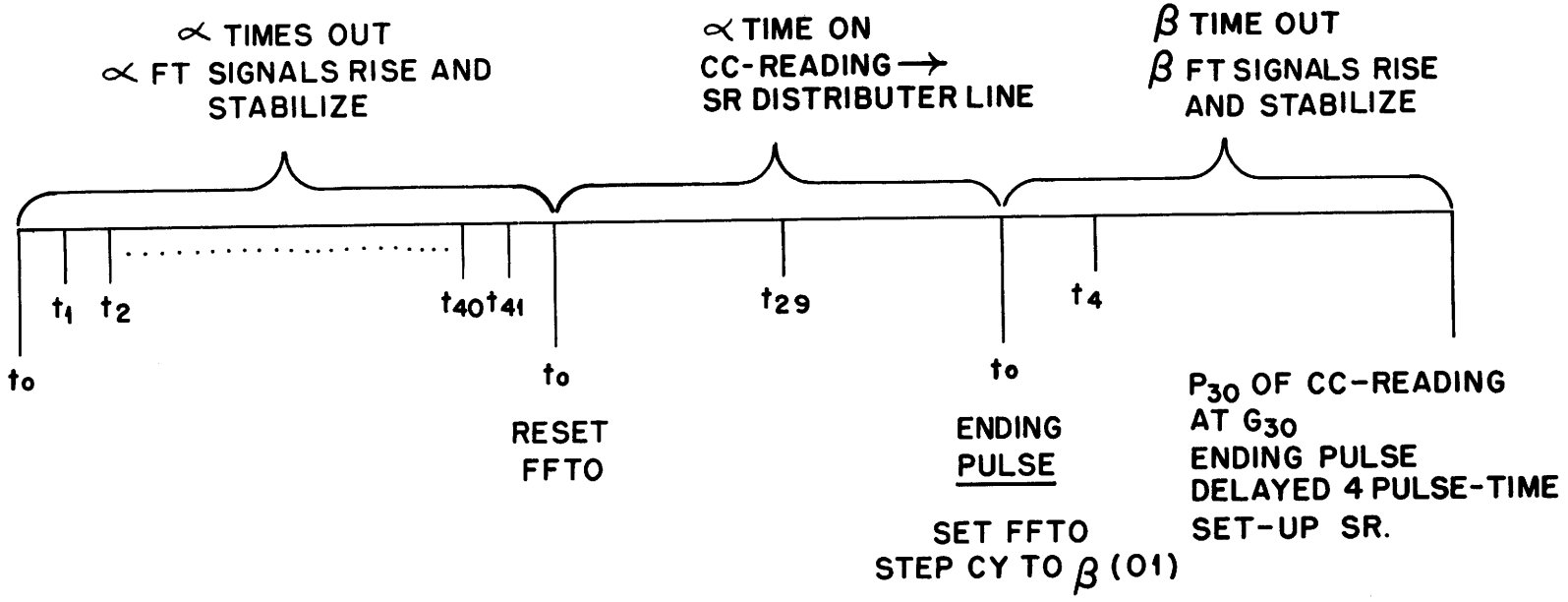
There are three logical steps necessary for the computer to carry out an instruction. All instructions are stored in the main memory; hence, the computer first obtains the address of the next instruction to be executed. Second, it makes use of this address to obtain the proper word from the memory and stores the $p_{30} \ldots p_{36}$ portion in a static memory. Third, the output of the static memory is interpreted by a function table which developes a pattern of signals (unique for each instruction) to enable the computer to carry out the indicated operation. This three stage cycle is repeated for each order executed. The three stages of this basic cycle are called by the Greek letters α, β, γ. The cycle counter CY, a two-stage binary counter, which counts 00, 01, 10 and then resets to 00, is used to remember which stage of the basic cycle the computer is on. See Chart B.

CY - 2	CY - 1	
0	0	= α
0	1	= β
1	0	= γ
0	0	= α

Thus, CY provides the input to a decoding function table whose output lines are the α line, β line and γ line. Of course, only one output line is activated at a time; which one is picked-up depends upon the reading of CY.

Description of the Basic Cycle of Operation

Alpha time: The memory address of the next instruction to be executed is stored in dynamic form in the control counter (CC). This register stores a full computer word (42 pulse positions), although only $p_{30} \ldots p_{33}$ are used to store the address. Since only one pulse position of a word is available to any instant of time on the recirculation line of a register, it is necessary to transfer the word in CC into a distributor line and convert the desired pulse positions into static (flip-flop) storage. When this is done, these pulse positions are available for as long as needed and



TIMING DIAGRAMS FOR ∝-TIME

FIG. 36

can be used to drive function tables which will in turn develope signals to extract the proper word from the memory. Hence, the operation consists of transferring the word in CC to a distributor line (without clearing CC) and converting the $p_{30}...p_{36}$ pulses to static storage. The seven flip-flops (FF30...FF36) are called the static register (SR).

When CY - 1 and CY - 2 are both in the zero state, the α line of the cycle counter decoding function table is picked up. This in turn enters the main encoding function table to select the 8, 9 and 10 lines. These signals are sufficient to effect the transfer from CC to SR.

The minor cycle in which CY reads 00 and FFTO is set is the α TO minor cycle. (See figure 36) It is during this period that FT signals 8, 9 and 10 are picked up and stabilized. By the end of TO, the αFT signals are alerting all gates to which they are applied. A t_0 passes gate 25 which is alerted by the set output of FFTO to reset the flip-flop. The time-out inhibition is now removed from all gates. FT signal 8 alerts the read-out gate of CC, G8, to allow the word contained in the register to pass on to HSB1A. The word began passing G8 as soon as FT signal 8 became permissive, but was prevented from reaching any other component by the TO inhibition in the high speed line amplifier (HSBA). HSBA consists of the PFR, G9A and G9B and the D1 delay in the output of G9A. FT signal 9 is present and when FFTO is reset G9A is alerted while G9B is blocked. Hence, the CC reading is switched from HSB1A to HSB2A.

The word left CC with the reference timing $p_0 = t_{41}$, but the HSBA inserts the necessary delay to allow it to enter HSB2A with the proper reference timing $p_0 = t_1$. It passes from HSB2A into the distributor line of the static register.

It is now necessary to consider at what time p_{30} will be applied to G30 which controls FF30. When p_{30} is applied to this gate, p_{31} will certainly be applied to G31 gate, etc., hence, the portion of the word that we wish to convert to static storage will be contained in the distributor line, when p_{30} is applied to G30 and it is at this time that the series of gates, G30, G31...G36 must be sampled.

The reference timing on HSB2A which feeds the SR distributor is $p_0 = t_1$. If a p_0 enters the delay line at t_1 then p_{30} enters at t_{31}. The p_{30} pulse will be applied to G30 fifteen pulses times later (after passing the delays of the distributor line) or at t_{46} or time t_4 of the next minor cycle.

FT signal 10 is also present during time-on (T.ON) and alerts G10. G10 has the same t_0 cycling unit signal as G25, in the TO circuit but the t_0 which resets FFTO at the beginning of T.ON does not pass G10 even though FT signal 10 is permissive, since FFTO does not charge state quickly enough to remove the inhibition on G10 before time t_1. Hence, it is a t_0 one minor cycle later which passes G10. The t_0 pulse output of this gate is called the ending pulse (EP), since it occurs at the end of each state of computer operation. This EP steps CY to 01 and sets FFTO and the computer is now in β TO. It clears the program counter (PC) to zero. (The use of this counter will be explained later). It resets all of the static register flip-flops clearing any information previously set up in static storage.

The EP also enters D4 delay and emerges at time t_4 of the β TO cycle. But this is the minor cycle following T.ON and it has been shown that it is at this time that the p_{30} ... p_{36} pulses of the CC reading are contained in the distributor line. Hence, the EP delayed four pulse times sample G30A, G31A...G36A at t_4 to set-up the memory address from CC in the static register.

Beta time: On β time the next instruction word is transferred from the memory to the static register. The task of reading into or from a memory channel is more complex than reading into or out of a one-word register such as CC. Since the recirculation time for a word in CC is one minor cycle, the word is available for read-out in every minor cycle. However, the recirculation time for a word in a memory channel is four minor cycles, and is available for read-out in only one out of each four minor cycles. Hence, to obtain a particular word from the memory necessitates, first, selecting the channel in which the word is stored and, second, selecting the minor cycle in which the desired

word is passing the read-out and read-in gates. The former, a positional reference, is called channel selection and the latter, a time reference, is called time selection.

Channel Selection - The first two digits of the memory address are stored in FF32 and FF33 of the static register. The output lines of these flip-flops drive a decoding function table which excites one of the lines C_0, C_1, C_2 or C_3. The excited line alerts the control gate, GT, of the channel indicated by the first two digits of the address. The channel selector signal will be present, of course, a few μsecs. after the address is set up in SR.

Time Selection - The time selection counter (TSC) is a two-stage binary counter which is stepped once per minor cycle by a t_2 from the cycling unit. The reading of TSC at any instant of time indicates which word is passing the read-out gate of a memory channel. It must be remembered that information in all memory channels circulates synchronously. Thus, when TSC reads 01 it indicates that the 01 word in all memory channels (that is, words 0001, 0101, 1001 and 1101) are passing the read-out gates of their respective channels.

To select the proper minor cycle for reading into or out of a memory channel, it is necessary to compare the two least significant digits of the address stored in FF30 and FF31 of SR with the TSC reading. And when the two agree, to produce a signal for one minor cycle which will alert all control gates, GT, in the memory.

The comparison of the word number with the TSC is accomplished by gates, G70, G71, G72, and G73. As long as the word number stored in SR does not agree with TSC, there will be a signal output from one or more of the comparison gates. However, when coincidence is reached, there will be no output from any of the gates. The buffed output line of the comparison gates is sampled by means of gate, G1A, to determine when the no signal condition, representing coincidence of TSC and the word number stored in FF30 and FF31, has been reached.

Gate, G1A, is sampled each minor cycle by a t_0 from the cycling unit, but the sampling pulse will pass G1A only when the inhibition from the comparison gate is absent. Hence, the t_0 will pass only when TSC agrees with the word number. This sampling is effective only on operations involving the memory, since only on these operations will FT signal 1 be present, which is necessary to alert G1A.

When a t_0 passes gate G1A, it sets the time selection flip-flop (FFTS) to produce the time selection signal (TS) for one minor cycle. The TS signal is limited to one minor cycle, because the TS signal alerts gate G1B, which passes the following t_0 to reset FFTS.

The TS signal alerts all control gates G_T, in the memory channel and in addition is applied to gates G3 and G5. If FT signal 3 is present, TS passes G3 to become the TS_0 signal which is used on reading out of the memory. If FT signal 5 is present, TS passes G_5 to become the TS_1 signal used on reading into the memory.

Although the TS signal is applied to the control gates of all memory channels, only one gate G_T will develop an output signal because only one of these gates will have a channel selector signal. Thus, if the address of the desired word is 1001 only the control gate of channel 10 will develop an output during the TS minor cycle.

The output of this control gate will alert the read-out gate G_0, the read-in gate G_1 and the clear control gate G_R of channel 10. If the operation is to transfer a word from the memory, the TS_0 signal will be present and gate G_0 will be fully alerted. This permits one word to leave the memory channel and enter HSB1M. If the operation is a transfer to the memory, TS_1 will be present and G_1 and G_C will be fully alerted. This permits one word to enter the memory channel from HSB2M through G_1 and blocks the recirculation path of the channel for one word time by inhibiting the clear gate, G_C, with the output of G_R.

It should be noted on operations involving the memory, that there may be several minor cycles of time-on during which the computer does nothing but wait for agreement of the TSC and the word number stored in the SR. These minor cycles of time-on are called latency time.

It is now possible to consider the operation of the computer on β time. The ending pulse produced at the conclusion of α time stepped CY to 01 (β) and set FFTO. During this β time out minor cycle, the cycle counter decoding function table decodes the 01 output of CY and excites the β line. This in turn enters the main encoding function table to select FT signals 1, 3, 7 and 11. Also on this TO minor cycle, the channel selector signal is picked up according to the most significant digit of the address stored in FF32 and FF33 of the static register.

On β time-on the TS signal is produced as soon as TSC agrees with the word number stored in SR because FT signal 1 is present. FT signal 3 is also present and the TS_0 signal is produced as well. This two signals are sufficient to read the desired word from the memory to HSB1M. The word leaves the memory with the reference timing $p_0 = t_0$ and after passing the PFR in HSB1M reaches HSB2A with the timing $p_0 = t_1$. From HSB2A the word is fed into the SR distributor line. (It is also sent to the read-in gates of the one-word register, but this is trivial since none of these gates are open). It is the $p_{30} \ldots p_{36}$ position of the word (the instruction portion) which are to be converted to static storage. The set-up time then is the same as for α time or at time t_4 following the time selection minor cycle. The ending pulse for the β operation occurs at the end of the β time selection minor cycle. It is obtained through gate G_7. This gate requires FT signal 7, present on the β operation, and the TS signal. The TS signal is necessary because the duration of β time-on varies depending upon the number of minor cycles of latency time. A t_0 passes this gate at the end of the β time selection minor cycle and steps the cycle counter to γ (10), sets FFTO, and resets the SR flip-flops. Delayed four pulse times, it samples gates G30...G36 or a t_4 and converts the $p_{30} \ldots p_{36}$ position of the word obtained from the memory to static storage for use on α time.

During β time selection, simultaneous with the transfer of the next instruction word from the memory to SR, the control counter reading is sent to the adder, advanced by 1 in the p_{30} position and returned to CC.

Gate G11 is opened during the β time selection minor cycle by the presence of FT signal 11 and the TS signal. The CC reading is transferred through G11D, a PFR and gate G58A to become one input to the adder. Gate G58A is open since neither the S_2 nor TO signals are present. The other input to the adder is a pulse (binary one) during the time the p_{30} position of CC is entering the adder. This is obtained through gate G11E which is alerted by FT signal 11 and TS, by passing a t_{29} from the cycling unit. The reference timing for the CC input to the adder is $p_0 = t_{41}$; hence, $p_{30} = t_{29}$. Thus, the t_{29} passed by G11E is added into the p_{30} position of CC. The result of this operation is to advance by one the address stored in $p_{30} \ldots p_{33}$ of CC.

The sum which is the new CC reading is returned to CC through a PFR and D1 on the sum output line of the adder and through read-in gate G11B of CC which is opened by FT signal 11 and TS. These same signals operate gate G11A where the output inhibits clear gate G11C in the recirculation path of CC to clear the register of its former contents.

Thus, at the end of a current β cycle, the address stored by CC is that which is to be used on the next β cycle.

Gamma time: On γ time the computer executes the instruction that was brought from the memory and converted to static storage during β time. The instruction code stored in FF34, FF35 and FF36 drive a decoding function table to pick up one of the eight possible instruction lines, which feed the main encoding function table. The pattern of signals produced by an instruction line and the address (if the memory is involved) are sufficient to carry out the indicated operation.

The instruction lines can be excited only on γ time since all instruction lines require the γ line from the cycle counter decoding function table.

Sec. 7. Introduction to the Operation of UNIVAC

Preliminary Discussion

A complete discussion of the operation of UNIVAC requires a description of the four-stage cycle of operations and detailed analysis of each of the forty-five orders. However, the remainder of this chapter will contain only a brief description of the four-stage cycle of operation.

Before covering a analysis of the mode of operation, we shall briefly review the word composition and the organization of the memory. A UNIVAC word is the fundamental unit of memory and consists of twelve decimal digits, where decimal digit is interpreted as any one of the 63 characters shown in the C-10 code. The digit positions in a word are numbered from left to right:

X X X X X X X X X X X X

1 2 3 4 5 6 7 8 9 10 11 12

Each decimal digit is composed of seven binary digits, grouped for convenience in discussion as shown:

X XX XXXX -one decimal digit-

the right four being the binary part, the center group of two known as the zone indicators, and the left most digit, the check pulse.

The C-10 code depicts the binary composition of each of the 63 valid UNIVAC digits. In particular, it is seen that the numbers zero through nine have binary zeros in the zone positions, while their binary part is in the excess-three notation mentioned in Chapter 2. The number of binary ones present in the binary part and zone indicators determines the check pulse. Thus, each decimal digit consists of an odd number of binary ones.

The main memory of the UNIVAC is a set of 100 acoustic delay lines called channels. Each channel has a capacity of 10 words. In addition, there is a ten word channel, register Y, a two word register V, and six one word registers A, X, L, F, CC, and CR. (See EBU-100 - Chart I in the Appendix) In the input circuitry there are six ten word channels known as register I, and similarly six ten word channels, register O, in the output section of the computer. The essential components of each channel and register are identical with the acoustic memory of the small computer described in the preceeding sections of this chapter.

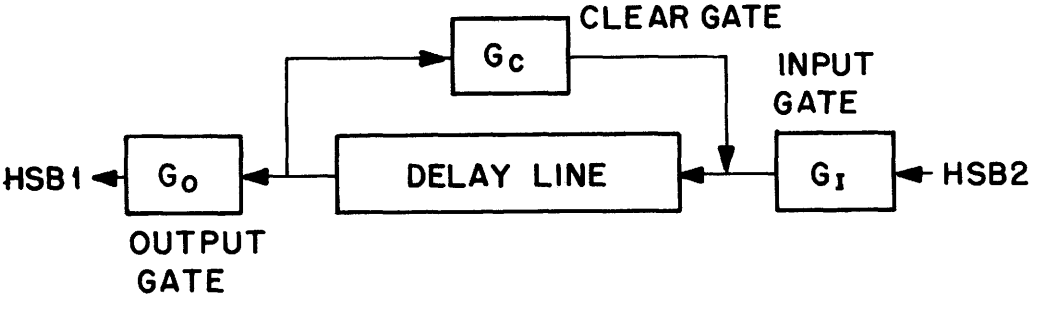


Fig. 37

The time necessary for the voltage train representing a complete word to pass a given point in the computer is known as a minor cycle, and is a fixed interval of time. The cycling unit (CU) emits signals marking the elapse of each minor cycle and generates the timing signals needed for the proper operation of all units.

The word positions in each ten word channel are numbered from zero to nine. The cycling unit assures the synchronization of all ten word channels, so that when word 5 of channel 00 is ready to emerge from its delay line, word 5 of each and every 10 word channel is ready to emerge from its respective delay line. Further, when digit position 12 is ready to emerge from register A, digit position 12 of some word in every delay line is ready to emerge. Words travel through the computer least significant digit first, that is digit position 12, first, followed by 11, 10, --- 1. Also, with each decimal digit the binary part preceeds the zones which preceed the check pulse.

Since the only time a word may be brought out of the memory is when it is passing by the output gate some means must exist for determining when the desired word may be operated upon. In order to identify word 073 it is not enough to specify that the word is in channel 07. We must know at what time the seventh word in the channel is ready to emerge. The time selection counter (TSC) counts the number of minor cycles elapsed modulo ten. Thus when TSC reads 3 the third word of each ten word channel is ready to emerge from its delay line. The time selection counter and the devices for selecting a particular channel of the main memory are called the memory switch.

The Four Stage Cycle of Operation

The normal operation of the computer involves four distinct stages called α, β, γ, δ, and are executed in that order. The cycle counter CY is a two stage binary counter (four stable stages 00 = α, 01 = β, 10 = γ, 11 = δ) which indicates the current stage the computer is on. Each stage is further divided by "time-out" periods, TO, as follows:

CY Readings	Stages
	α TO
00	α
	β TO
01	β
	γ TO
10	γ
	δ TO
11	δ

The purpose of these several divisions will become clear as the characteristics of each stage are considered. Also, in the discussion to follow, it will be helpful to the reader to refer to the 'Simplified Block Diagram of UNIVAC' (EBU-100) - Chart I in the Appendix.

To begin, assume that CY is in the α stage and the time out flip-flop is set, producing the α TO period. When CY reads α (00) it causes the function table to generate certain signals associated with the α stage. The presence of the time out signal which lasts for one minor cycle prevents any further action of the computer and is necessary to allow the function table signals to rise to their normal voltage level. Thus one minor cycle elapses and then a pulse from the cycling unit removes the time-out signal by resetting flip-flop TO. Removal of TO allows the α function table signals to open the output gate of the control counter CC whence the word in CC passes out onto high speed bus IA, (HSB1A) into the high speed bus amplifier (HSBA) which under the action of the α function table signals switches the word onto HSB2A. While the word is on HSB2A it is examined by the HSB odd-even checker (HSB OEC). As each decimal digit enters HSB2A the number of binary ones is counted by OEC. If for any digit an even count is registered, an error signal is produced stopping the computer on the next time out period. The word from CC enters HSB2A and enters the control register CR whose input gate has been opened by an α FT signal. It passes through CR and enters the static register SR. SR consists of a set of flip-flops and at the proper time a pulse from CU transfers the digits 7 through 12 from their dynamic form as a voltage train into static form in the SR flip-flops. At the same time a pulse from CU is allowed to step CY to β (01) and set FFTO. The word in CC will appear as:

000000000XXX

where XXX is a number between 000 and 999 and thus 000XXX is set up in SR at the end of α time.

When CY was stepped to β (01) the β function table signals are generated. Again the presence of TO prevents any action of the computer for 1 minor cycle in order that the FT signals have time to rise to full strength. During each minor cycle the reading of TSC and the right hand digit (time selection digit) in the static register are compared.

During the minor cycle in which they agree, a flip-flop (FFTS) is set and in conjunction with the β FT signals a signal is applied to the output gate of each of the 100 channels of the main memory. Meanwhile digits 2 and 3 in SR have been decoded in the channel selection part of the memory switch and a signal is applied to the output gate of one particular channel; thus, only one channel during one minor cycle will have both signals present to open its output gate. This minor cycle is called the time selection minor cycle and may be as much as the 10th minor cycle after CY was stepped to β. The TS and β FT signals close the clear gate and open the input gate of CR. The word from the selected channel passes out onto HSB1M into the HSB amplifier where it is routed onto HSB2A. It is also examined by HSB OEC. From HSB2A the word enters CR. During this TS minor cycle the output gate and a special input gate of CC are opened while the clear gate is closed. The contents of CC are routed to the adder by a special path where 000 000 000 001, generated by CU, is added to it and the sum coming from the adder returns to CC. At the termination of the TS minor cycle CY is stepped to γ and TO set. The presence of TO closes the output gate of CC, the input gate of CR and opens the clear gates of CC and CR. The input gate of CC is closed somewhat later in order that all of the sum returning from the adder will enter CC.

Returning to the β TS minor cycle, the word from the memory enters the recirculation path of CR and it is also routed into the SR. At the same time, the ending pulse from CU which sets TO and steps CY will also set up the digit positions 1 - 6 in the SR flip-flops.

Recapitulating:

α TO: α function table signals rise to strength.
α: digit positions 7 - 12 of CC set up in SR. Ending pulse from CU steps CY to β and sets TO.

β TO: β function table signals rise to strength.
β: during β TO and each following minor cycle digit position 12 (TS digit) in SR is compared with TSC. Upon agreement the word in the memory location specified by digits 10 - 12 of CC (which

was set up in SR during α) is transferred to CR and the left 6 digits (left instruction) transferred to SR. The contents of CC are augmented by 1. The ending pulse for CU steps CY to γ and sets TO.

Attention is called to the similarity of stages α and β in UNIVAC to that of the elementary computer. However, the small computer has no device similar to CR, the instruction word coming from the memory entering SR directly. Since a UNIVAC instruction is completely defined in six digits, it permits two instructions per word. In order to speed computation, the number of memory "look-ups" should be reduced to a minimum. Thus by use of CR, it is possible to extract two instructions from the memory at one time. The instruction pair is stored in CR during β time, and the left hand instruction (digits 1 - 6) are sent to SR in time for the ending pulse that steps CY to γ and sets FFTO to set-up the instruction in the SR flip-flops.

During γ TO, the instruction in the SR actuates the function table directly to produce the signals peculiar to that instruction. After TO is removed the instruction is executed. CY, being on γ, produces a special function table signal that passes the ending pulse of the left-hand instruction to set-up the right-hand instruction in SR at the same time as CY is stepped to δ and TO set. This same process is then repeated for δ time. However, the ending pulse now steps CY to α and the entire four steps are repeated.

This completes the discussion on the logic and operation of UNIVAC and the components used by UNIVAC as planned for this manual. This, of course, is an incomplete discussion of the subject and it is planned, at some later date, to issue a complete manual on the logic of UNIVAC.

Chapter 10

Flow Charts - An Aid to Programming

SEC. I. PRELIMINARY DISCUSSION

In the previous chapters were discussed the forty-odd orders which the UNIVAC will execute. With these orders were illustrative examples designed to clarify each order and to suggest how these basic operations may be organized to effect the solution of a problem. The purpose of this chapter is to examine further the methods of organizing these instructions to accomplish desired results.

From a logical standpoint UNIVAC instructions (and those of most digital computers) can be grouped into three categories, called Logical Operations:

A) Transfer of information from one storage location (or medium) to another. In this category are the UNIVAC orders B, C, E, F, G, H, J, K, L, R, V, W, Y, Z, 10, 50, 1n, 2n, 3n, 4n, 5n, 6n, 7n, 8n.

B) Counting and its mathematical variant of algebraic addition, multiplication, and division; comprising the A, D, S, M, N, P, X, On, -n, .n, and ;n orders.

C) Choice between which of two sets of instructions is to be performed. In this group are the Tn, Qn, 00, U, 90, Overflow, and ,0 orders.

"Programming" may thus be defined as the combining of the three Logical Operations to "solve" a particular problem. A "program" is, then, the actual combination of such operations.

There is no general method available, as yet, by which one can determine how the Logical Operations may be assembled to solve a problem, or even to determine if the problem has a solution suitable for digital computers. This is, therefore, a question that must be decided on the basis of individual experience and judgement. Again, to choose among several proposed methods, experience and judgement are the best aids in determining which is most suitable.

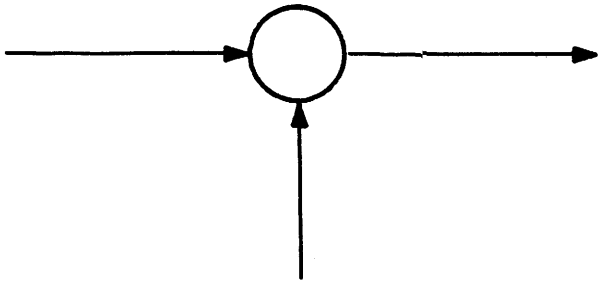
However, once the general plan of attack has been formulated there are aids to the programmer in accomplishing the yet considerable task of putting the solution in terms of the actual instructions the computer can carry out. Of these aids we shall discuss only one --- flow-charts. Of the others, short-order code, executive routines, tape libraries, the reader is referred to the reference in the bibliography.

Flow-charting is a graphical device to aid the programmer in the detailed organization and improvement of the basic plan of attack. Essentially it consists of symbols for the Logical Operations we have discussed and devices to assemble and order them.

SEC. 2. ASSEMBLING AND ORDERING SYMBOLS

The "path of computational flow" is indicated by a directed line segment:

The inference is obvious that the next step in the problem will be found in the direction of the arrow. Where two or more different paths of computational flow merge to follow one common path a "fixed connector" is placed at the point of merging:



By numbering the fixed connectors we need not indicate a merging of flow paths by the actual joining of the lines as shown above. This is especially advantageous where the merging flow lines would have to come from widely separated areas of the chart. Thus, figures A and B are identical operations.

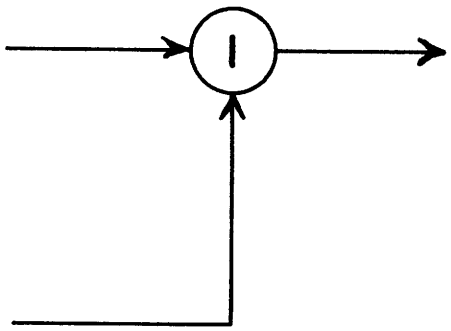


Figure A

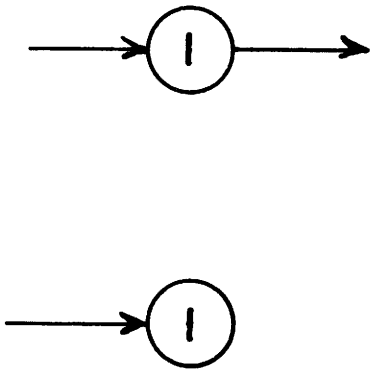
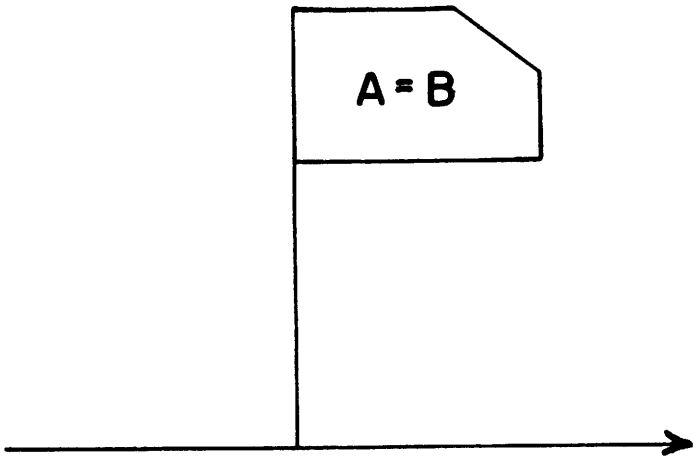


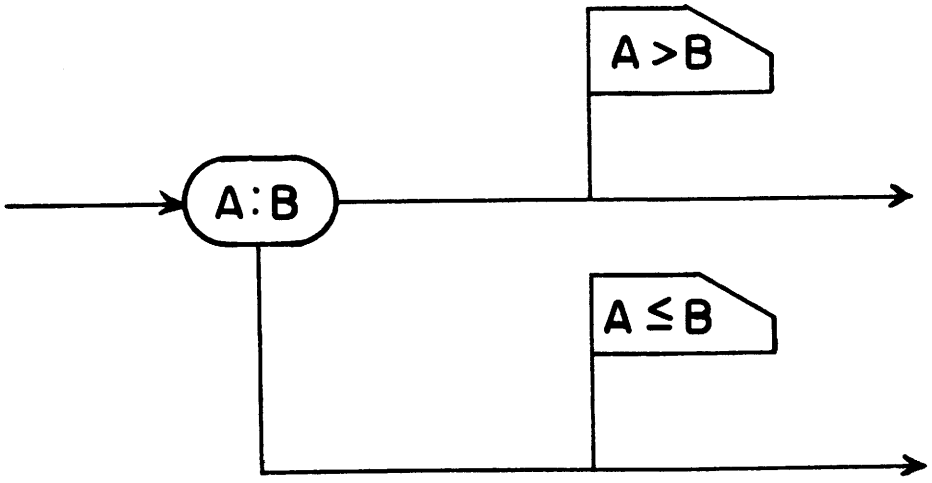
Figure B

If for reason of clarity we wish to indicate that a certain condition is true at a certain point in the line of computational flow, a "flag" asserting the condition is attached to the flow line:

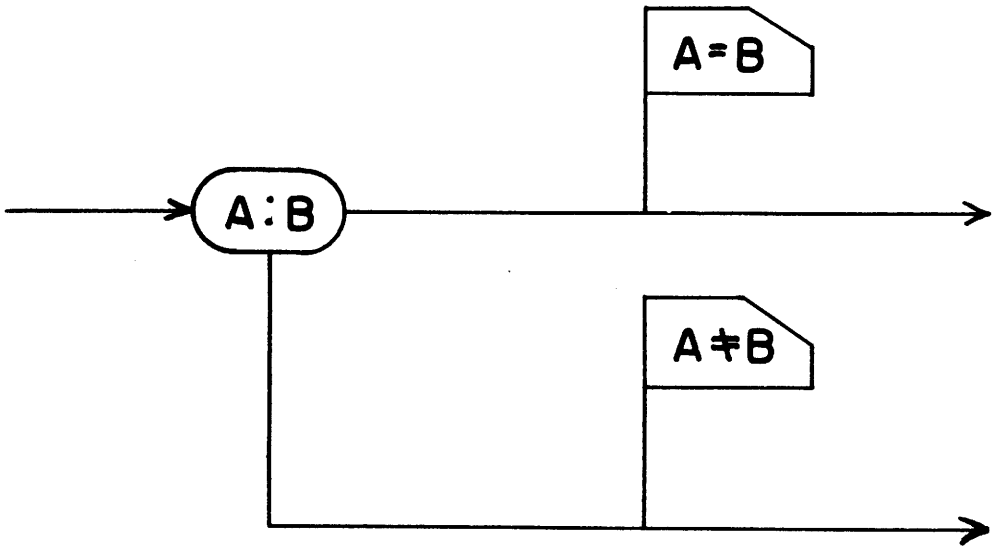


SEC. 3. LOGICAL CHOICE

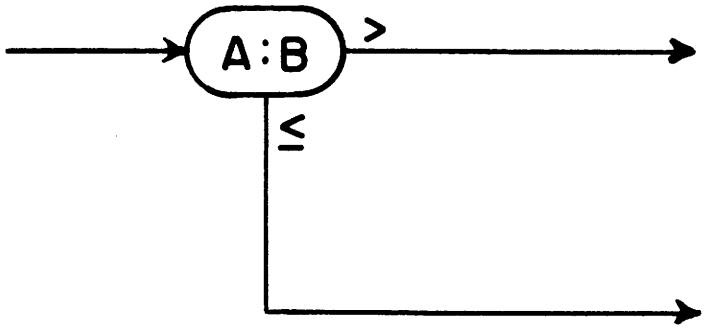
Logical choice between two paths of computational flow based upon the relative magnitudes of two quantities A and B is indicated by:



Or, for the choice of paths based upon the equality of two quantities:



Standard practice has been to dispense with the flags in the case of logical choice by the following scheme:

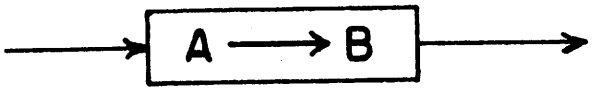


Ambiguity is avoided if we remember that the condition existing on either outward flow line is that obtained by replacing the colon (:) with the > or ≤ symbol where the choice is made on relative magnitude or by = or ≠ when the choice is based upon equality.

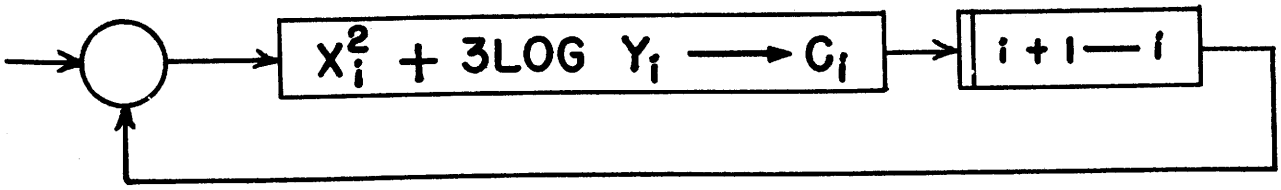
SEC. 4. COMPUTATION AND TRANSFER SYMBOLS

The evaluation of a formula or straight computation is indicated inside a rectangular box:

Where the arrow indicates that C_i is now the quantity $x_i^2 + 3 \log y_i$. This box thus means a computation and a transfer operation. The redundant case of a transfer only would appear as:



A special case is made for all arithmetic operations which are essentially counting in nature. For example, if we desired to compute another C_i we would want to increase the subscript i. This is what we call a counting operation and is indicated by a "substitution" box:



The subscript idea will be covered more completely later.

SEC. 5. ILLUSTRATIONS SHOWING USE OF SYMBOLS

We shall introduce other commonly used flow-chart symbols when needed, but first let us see how these symbols may be combined to form a program.

Example 1: Given a set of thirteen cards, determine if a jack is present.

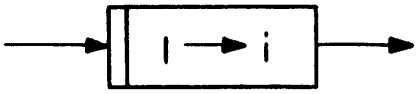
The analysis of this problem is quite simple and straight-forward: let us examine the face value of each card in turn. If the card is a jack, we shall indicate in some manner that the problem answer is "yes". But if no jack turns up in any of the thirteen cards, we will give the answer "no". A question arises as to the way we shall indicate that we are considering a particular card of the set of thirteen cards, and how we can make sure that our examination will overlook no cards. The simplest reply is that such a device already exists. Let us assume that the cards are stacked -- we can always do this, then it is natural to refer to the top card of the stack as "the first card", the card immediately under this one as "the second card", and the next one down as "the third card", and so forth until the thirteenth card is reached which we call "the last card". As long as we do not alter the order of the cards by shuffling the deck, the "fourth card" is quite sufficient to designate one and only one card. Again, when we are examining the cards, if we start by looking at the top, or "first card", then at the "second card", and so on, each card in its turn, we know that when we have examined the "thirteenth card" we will have looked at each and every card, and will have missed none of them.

Assuming that the cards are now stacked we can draw a flow-chart of our solution at once, as shown in figure 1.

It is obvious that even for the small number of cards we have to examine this flow-chart is a long and cumbersome affair. To those with mathematical experience an improvement is easily evident. The great clarity and power of mathematical thought lies in its use of symbols as a shorthand notation for a word description which at best is often long

and ambiguous. Let us make use of this technique and see how it may aid us in condensing this flow-chart. If we designate the letter C to stand for card, then by introducing a subscript we can easily refer to any particular card: C_1 being the first card, C_2 the second card, C_3 the third card, and so forth. C_i is the obvious generalization of this concept, and will thus stand for the ith card of the thirteen. Initially we will set i = 1 so that C_i then means C_1 or the first card. After examining this card we can increase i by 1 and then C_i will mean the second card C_2. By testing for i = 13 we will be able to tell when we have examined the "last card". The flow-chart will now appear as shown in figure 2.

Let us take a brief tour through the flow-chart so that every operation shown will be clearly understood. From the start circle, the first operation encountered is the substitution box



which tells us that the card counter i is initially set to read 1, the first card. The comparison symbol

is next to be encountered. At this time, we examine the selected card to determine if it is a jack. Two paths of computational flow are shown leaving the comparison box. This is always the case; which path we follow depends, of course, upon the result of the comparison. If the card C_i is a jack, we will leave the comparison box along the "equal to" flow line and pass directly into the operation box instructing us to print the answer "yes". Having found a jack we then stop. However, if we take the "not equal" path from the above comparison box, we know that the ith card, C_i, is not a jack.

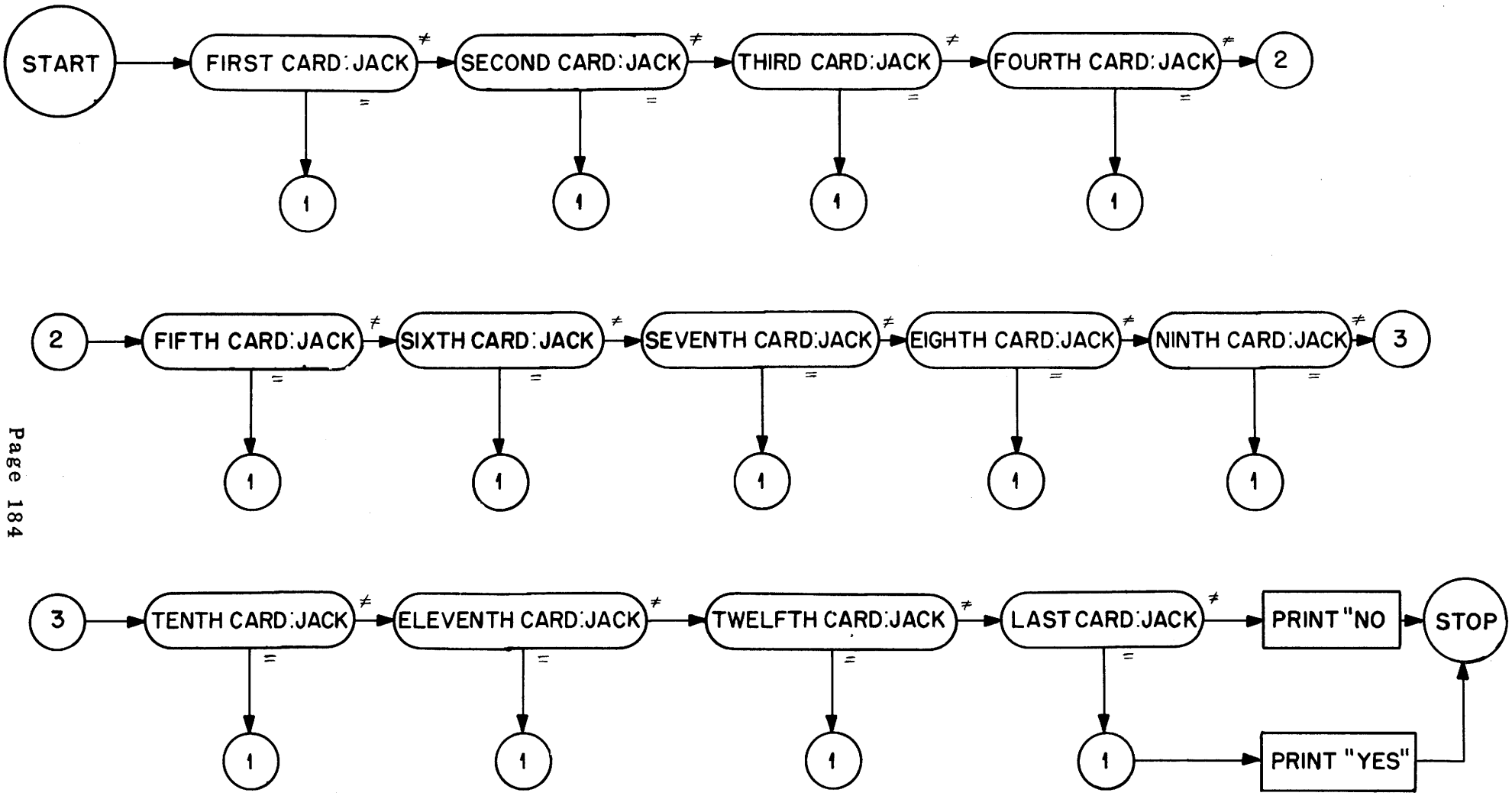

START
FIRST CARD:JACK
SECOND CARD:JACK
THIRD CARD:JACK
FOURTH CARD:JACK
2
1
2
FIFTH CARD:JACK
SIXTH CARD:JACK
SEVENTH CARD:JACK
EIGHTH CARD:JACK
NINTH CARD:JACK
3
3
TENTH CARD:JACK
ELEVENTH CARD:JACK
TWELFTH CARD:JACK
LAST CARD:JACK
PRINT "NO
STOP
PRINT "YES"

FIGURE 1

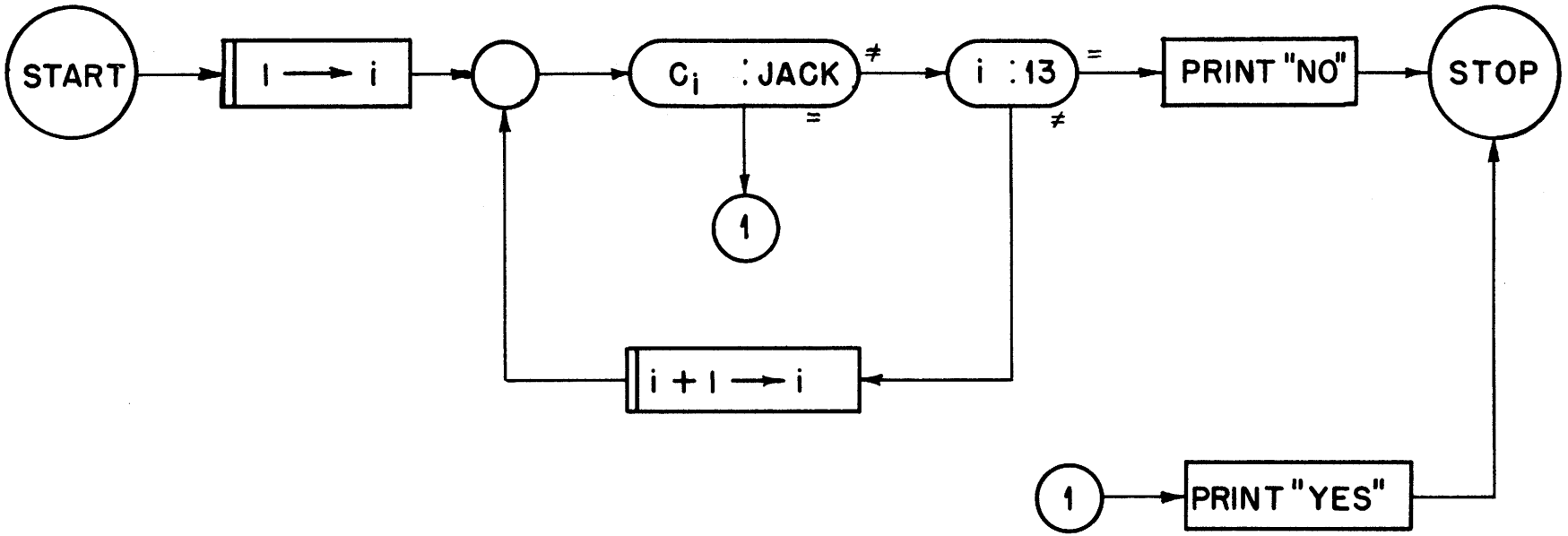
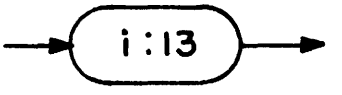


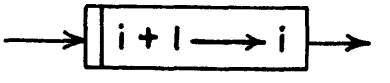
FIGURE 2

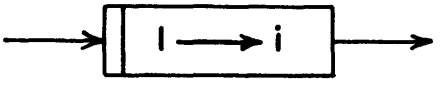
Therefore, we want to examine the next card. But first we must assure ourselves that there is a next card, that is, that the card we just examined was not the last card in the stack. The comparison box



will determine if the card just examined, C_i, is the last card, C_{13}. Thus, if i = 13, we enter the operation box,

telling us to print the answer "no" since we have looked at all thirteen cards and found no jack among them. If the card just examined was not the last card, we then increase the card counter i in the substitution box and proceed to examine this new card in the same fashion, bypassing the box.





The flow-chart is a precise description of how we intend to carry out the process. But before we can instruct the computer to solve this problem for use we must indicate what a card is to the UNIVAC. The pertinent information present on a playing card is only the suit and the face value. Let us use one UNIVAC word for each card, laid out in the following fashion: the first seven digits will contain the suit and the remaining five the face value of the card. Any digits not needed in either field will be filled by space symbols. Examples are given:

DIAMONDACE△△
HEART△△KING△
CLUB△△△QUEEN
SPADE△△JACK△
SPADE△△10△△△
.
.
.
DIAMOND2△△△△

We could just as easily have designated the suit and face value of the cards by numbers (which we shall do in the next problem) but for the present we shall use the notation just presented. If we store the cards in memory locations 101 to 113, then we can use the memory location of any given card as the subscript i of our analysis. To code this problem we need only assemble the few orders necessary to carry out each of the operations designated by the flow-chart. The coding for this problem begins on the following page.

```
000    [L 008                    Jack∆0 000000 --> rL
              B    101]          i th card, C1 --> rA
001    ;7 000                    Shift off suit
              Q    007          If C1 is a jack, transfer control
```

```
002    B  000                    i --> rA
              L    009
003    00 000
              Q    006          Transfer control if i = 13
```

```
004    A  010
              C    000          i + 1 --> i
005    00 000
              U    000          Transfer control to examine next
                                  card
```

```
006    50 011                    ŔNOT∆PRESENT --> Supervisory Con-
              90   000             trol Printer
```

```
007    50 012                    ŔPRESENTíííí --> S.C.P.
              90   000
```

```
008    JACK∆0
              000000
009    L00008
              B00113
010    000000
              000001
011    ŔNOT∆P
              RESENT
012    ŔPRESE
              NTíííí
```

A more complex problem is the following:

Example 2: Given thirteen playing cards, determine if they contain a royal flush (ten, jack, queen, king, and ace all of the same suit.

This problem is trivial for a computer rivaling the brain in complexity and coupled to an input servo as flexible as the eye. Having only a UNIVAC to work with however, we might approach the problem as follows:

Let us examine each card in turn to determine if it is a diamond. If it is, we then ask if it is a ten. If it is not a ten, we continue by examining the next card, but if it was a ten of diamonds, we then re-examine the entire set of cards to see if there is a jack of diamonds present, and so on until we find the ace of diamonds. (We start again from the first card because we may have passed over the jack while looking for the ten!) However, if one of the cards making up the royal flush in diamonds is missing, we start the process over again, this time concentrating upon hearts, terminating the process as soon as we find a royal flush in some suit or when we have examined all suits and found no royal flush.

As in the previous problem let the symbol C_I stand for the ith card of the thirteen cards. Since we are going to be concerned with the suit of each card as well as the face value let us further refine the symbology so that when we are examining the suit of the ith card we can specifically indicate this. Therefore, let the symbol C_I^S stand for the suit of the ith card and C_I represent the face value of this card. Also, to bypass the trouble of writing out "jack" or "diamonds", let F_2 stand for a deuce, F_3 a trey, ..., F_{10} a ten, F_{11} a jack, F_{12} a queen, F_{13} a king, and F_{14} the ace, while S_1 will mean the first suit -- diamonds, S_2 -- hearts, S_3 -- clubs, S_4 -- spades. Generalizing then, F_K will be the kth coded face value and S_J will be the jth suit. The complete flow-chart will now appear as shown in figure 3.

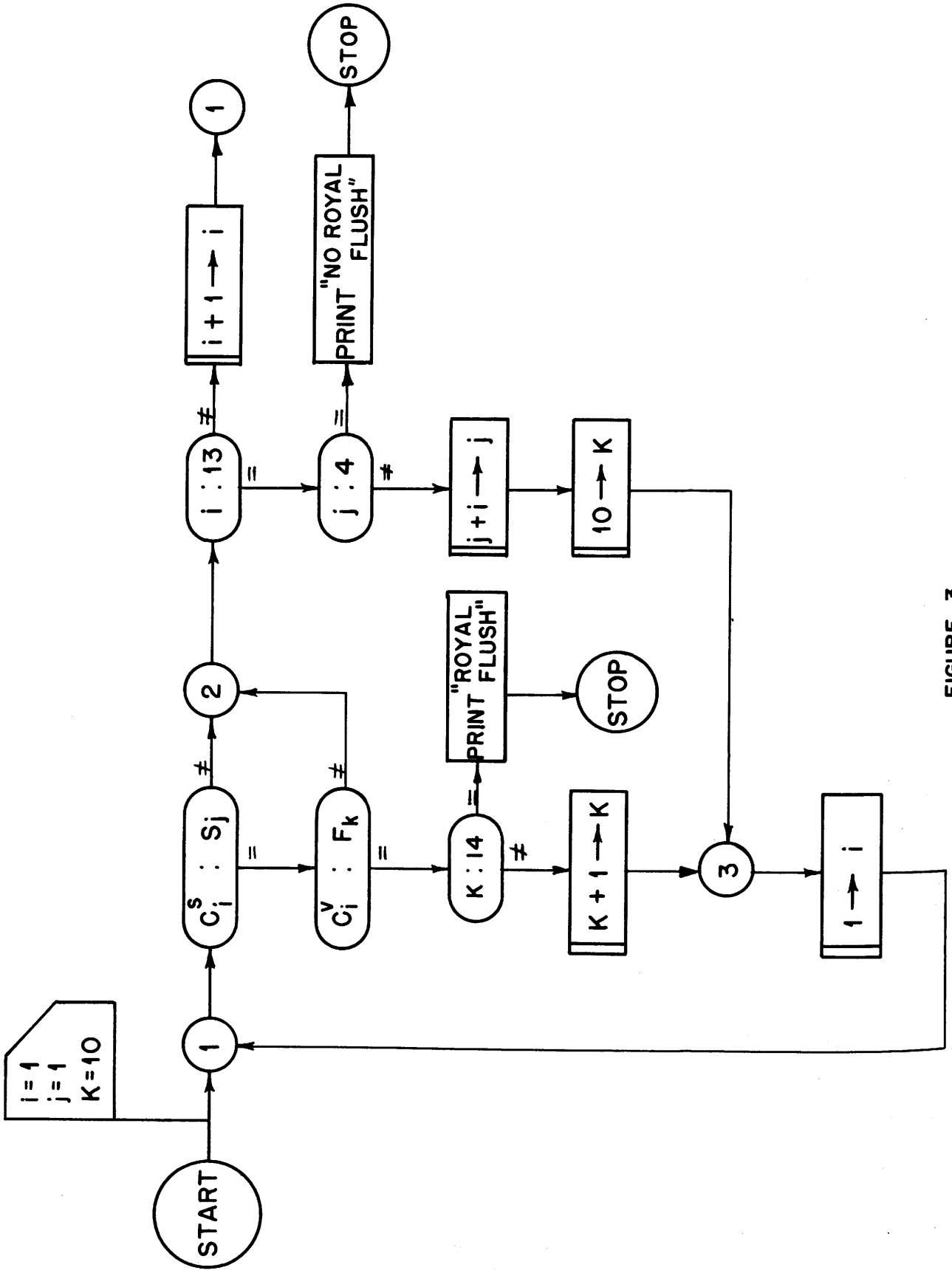
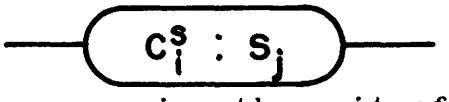
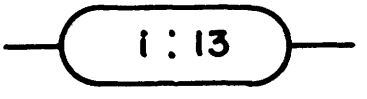

START
i = 1
j = 1
K = 10
1
C_i^S : Sj
C_i^V : F_k
K : 14
"ROYAL FLUSH"
STOP
K + 1 → K
3
1 → i
2
i : 13
j : 4
j + i → j
10 → K
i + 1 → i
1
"NO ROYAL FLUSH"
STOP

FIGURE 3

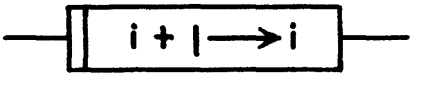
Again, we shall tour the flow-chart so that every operation shown will be clear. The flag to the right of the start circle tells us that initially the counter i reads 1, the first card of the thirteen, the counter j reads 1, so that S_j now stands for the first suit to be examined, diamonds. We have set k = 10 so that F_k means a ten. The first operation to be encountered is the test box.



where we examine the suit of the card C_i (C_1 for this first time through) to see if it is a diamond. If it is not a diamond, we next test the card counter i against 13:

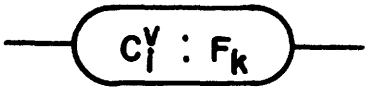


since this card is not the last card we increase the card counter by one



and go back to connector 

to examine the next card. If the card was a diamond, however, we then examine the face value to see if it is a ten. If it is not, we pass through the section that tests for the last card and advances the card counter. If the card was a ten of diamonds, we exit from the equal-to path of the test box

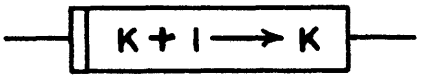


and into the box

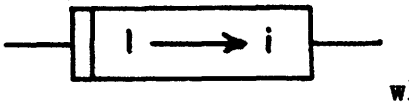
At this point we ask the question is the card value we found an ace. If it eventually becomes an ace, we know that we will have found a royal flush. Therefore, we enter the operation box

which prints ROYAL FLUSH

and then stops the search. Of course, for the first time through $F_K = F_{10}$; therefore, we leave the test along the not equal to path and into the box

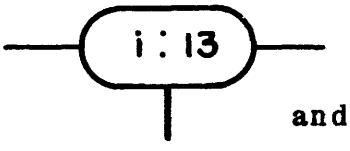


Here we increase k by 1 so the F_K now signifies the jack. Passing the unnumbered connector we enter the box

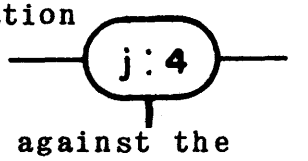


which resets the card counter so that we can re-examine the first card when we leave connector 

If we do not find a royal flush in the first suit, eventually we will pass through the equal-to side of the test box

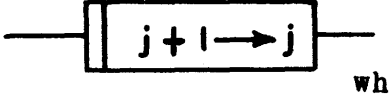


and encounter the operation

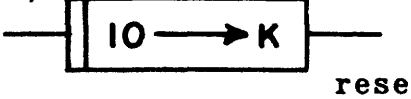


Here the current suit just examined is compared against the last suit, spades. If the suit is spades, it means that we have looked for a royal flush in all of the suits and having found none we enter the box

This indicates that there is no royal flush and we stop. However, if the suit just examined was not spades, we wish to examine the cards again for a royal flush in the next suit. This is done in the series of operations:



where we increase the suit counter j so that we test each card against the next suit,



reseting the card value counter

so that we first look for a ten in the new suit, and finally

where again we reset the card counter i so that we can re-examine the entire deck of cards.

This flow-chart is now the complete description of our solution to the problem. We have reduced the problem to a particular combination of counting and testing operations, which can be carried out by any computing device capable of executing the three Logical Operations.

For those readers not yet convinced, it should be point, ed out that the human brain is just such a computing device, and trying out the program with a few sample sets of cards will demonstrate and clarify the validity of the flow-chart.

It is probably evident to those mathematically inclined that the flow-chart which defines a computing process is a complex function itself whose independent variable is a set of thirteen cards and whose dependent variable has one of two possible values, "royal flush" or "no royal flush".

Preparation of the program is now a simple matter. We need only to specify what appearance the cards shall have in the UNIVAC and where they shall be located. Let us assume that each card will be represented by a word of the form:

0S0 000 000 0VV

where S is again the suit and will consist of a number, 1 for diamonds, 2 for hearts, 3 for clubs, and 4 for spades. VV is the face value of the card, duece being 02, trey = 03, through 10 for the ten, 11 for the jack, 12 the queen, 13 the king, and finally 14 for the ace. Further, let us assume that the cards are placed in memory locations 101 to 113. Thus, the memory location of each card will serve as the counter i of the flow chart.

Coding for Example #2

```
000    L    021
                  B   (101)
001    .2   000
                  Q   005      Transfer control if C_i^s = S_j
-------------------------------------------------------------

002    B    000
                  L   022
003    A    026
                  Q   014      Transfer control if i = 13
-------------------------------------------------------------

004    C    000                i + 1 --> i
                  U   000      Transfer control to examine
                                 next card
-------------------------------------------------------------

005    K    000
                  X   000
006    01   000
                  L   027
007    00   000
                  Q   009      Transfer control if C_i^v = F_k
-------------------------------------------------------------

008    00   000
                  U   002      Transfer control to increase i
-------------------------------------------------------------

009    B    027
                  A   028
010    L    029
                  Q   018      Transfer control if k = 4
-------------------------------------------------------------

011    C    027                k + 1 --> k
                  00  000
-------------------------------------------------------------

012    B    030
                  C   000      1 --> i
```

Example #2

013	00 000	U 000	Transfer control to examine cards again
014	B 021	A 031	
015	L 032	Q 019	Transfer control if j = 4
016	C 021	B 033	j + 1 --> j
017	C 027	U 012	10 --> k Transfer control to set i = 1
018	50 025	90 000	
019	50 023	50 024	
020	00 000	90 000	
021	[00 010	000000]	S_j
022	L00021	B00114	
023	RNOΔRO	YALΔFL	
024	USHiii	iiiiii	
025	RROYAL	ΔFLUSH	
026	000000	000001	
027	[000000	000100]	F_k

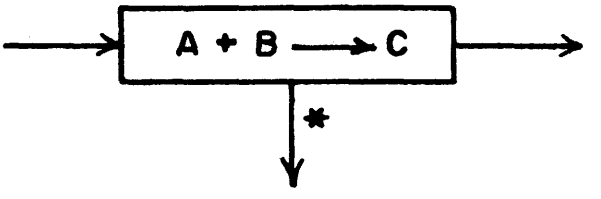
Example #2

```
028     000000
                000010
029     000000
                000150
030     L00021
                B00101
031     000010
                000000
032     000050
                000000
033     000000
                000100
```

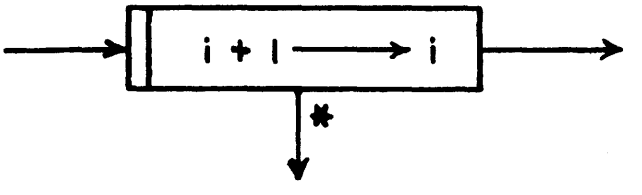
SEC. 6. ADDITIONAL FLOW CHART SYMBOLS

There remains three other important flow-chart symbols in common use:

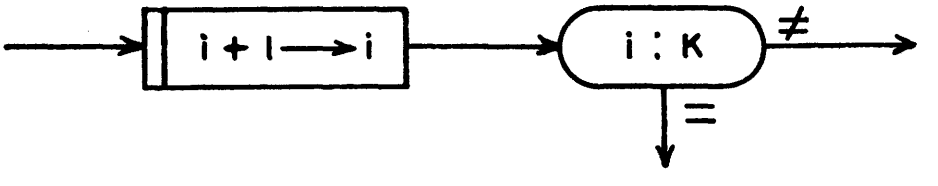
In some operations the possibility of an overflow is present, and when it occurs the problem may call for special operations to be performed. The starred (*) outward flow line is followed in the event of overflow occurring during the operation shown:



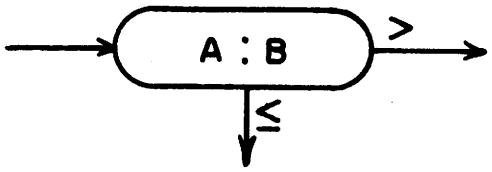
In UNIVAC the occurrence of overflow is often-times used as a test for a counter or variable line of coding reaching a particular value. The use of an overflow symbol as described above is not to be used in such cases. Besides the fact that a properly executed flow-chart is not to be interpreted as a picture of the coding but rather as a picture of the logical elements entering the thinking-out process the programmer undertakes in solving the problem, the use of the symbol in the following case.



leaves the reader of the flow-chart in doubt as to what value of 1 one takes the starred output line. If a discrimination on i is to be done it should be shown as:



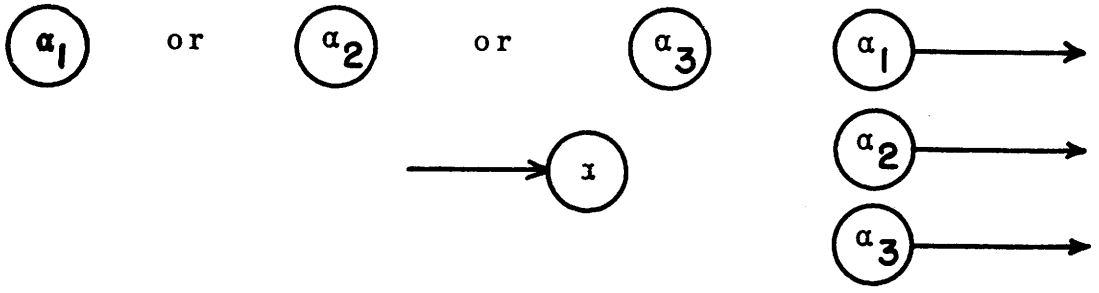
It cannot be over-emphasized that the flow-chart symbol:



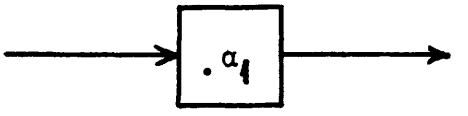
does not represent the UNIVAC T order, but does represent the logical need of discriminating between a set of operations to be performed by the relative sizes of the quantities A and B. The flow-chart symbols shown do not restrict the programmer from carrying out the indicated operation by a variety of instruction codes or combinations of codes.

We defined, earlier, the "fixed connector" as a symbol indicating the point of merging of separate flow lines. The "variable connector" is its counterpart. This symbol is placed in the line of computational flow where the common input splits to select one of many possible paths. Variable connectors always bear an identification symbol. Many symbols are in common use but in this chapter we shall always designate a variable connector by a letter of the Greek alphabet. In the figure shown below, the path of computational flow upon reaching the variable connector

will continue along one, and only one of the possible paths



The specification of which possible path will be taken must, of course, be stated before the variable connector is reached. Specifying which path is to be taken is called "setting" the connector, and is indicated by the square box:



By the use of variable connectors, a particular series of operations common to many different sections of the flowchart may be performed without repeating the operations for each section:

Example 3: A tape on UNISERVO 2 contains a series of three-word items arranged in ascending order. The number of items present is not known, but we do know that if the last item does not occur in the last three words of a block, the remaining words of the block consist of twelve Z's, called sentinels. In this case and the case where the last item does occur at the end of a block, two additional blocks will be found on the tape each word of which consists of twelve Z's. The first word of each item is a serial number and it is by this serial number that the items are ordered. The other two words contain quantities A and B as shown in the item layout:

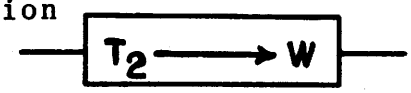
```
SSS SSS SSS SSS
000 000 AAA AA0
000 000 BBB BB0
```

All items of the same serial code are to be summed and the summaries are to be written on a blank tape mounted on UNISERVO 3 which will then be printed on the UNIPRINTER, producing a table such as:

SERIAL CODE	QUANTITY A	QUANTITY B
SSSSSSSSSSS	AAAAAAAA	BBBBBBBB
.	.	.
.	.	.
.	.	.

If we let the serial number of the ith item be represented by the symbol W^{i}_{0}, the quantity A by the symbol W^{i}_{1}, and the quantity B by W^{i}_{2} the flow-chart will appear as shown in figure 4.

The flow-chart has been divided into seven sections for convenience in discussion. In section 1, we encounter the operation



designating in this case the transfer of the first twenty items (one block) from the tape mounted on UNISERVO 2 into the twenty three-word items W^{1}... W^{20} denoted collectively by the symbol W. After transferring the serial number of the first item to become the quantity S we examine the block just brought in from tape to determine if it is a complete or partial block. If it is a partial block, we set connector

in order to examine each item of the block for the presence of a sentinel. Thus, by using the variable connector

we can eliminate testing each and every item from tape for a sentinel and instead test only one item per block, and only in the last block need we test all the items. Section 2 contains the main processing loop of the problem. Here each item serial code is compared against the serial codes of the current tallies. If it is equal, we obtain the new subtotals for D and E, examine the item counter i to see if all items for this block have been processed. If not, advance the item counter and return to process the new item. If, however, the items in the current block have all been processed, (i = 20), we read a new block of twenty items into the W storage, and examine this new block to see if it actually contains 20 items, that is, determine if it contains any sentinels. If it does contain sentinels, we set connector

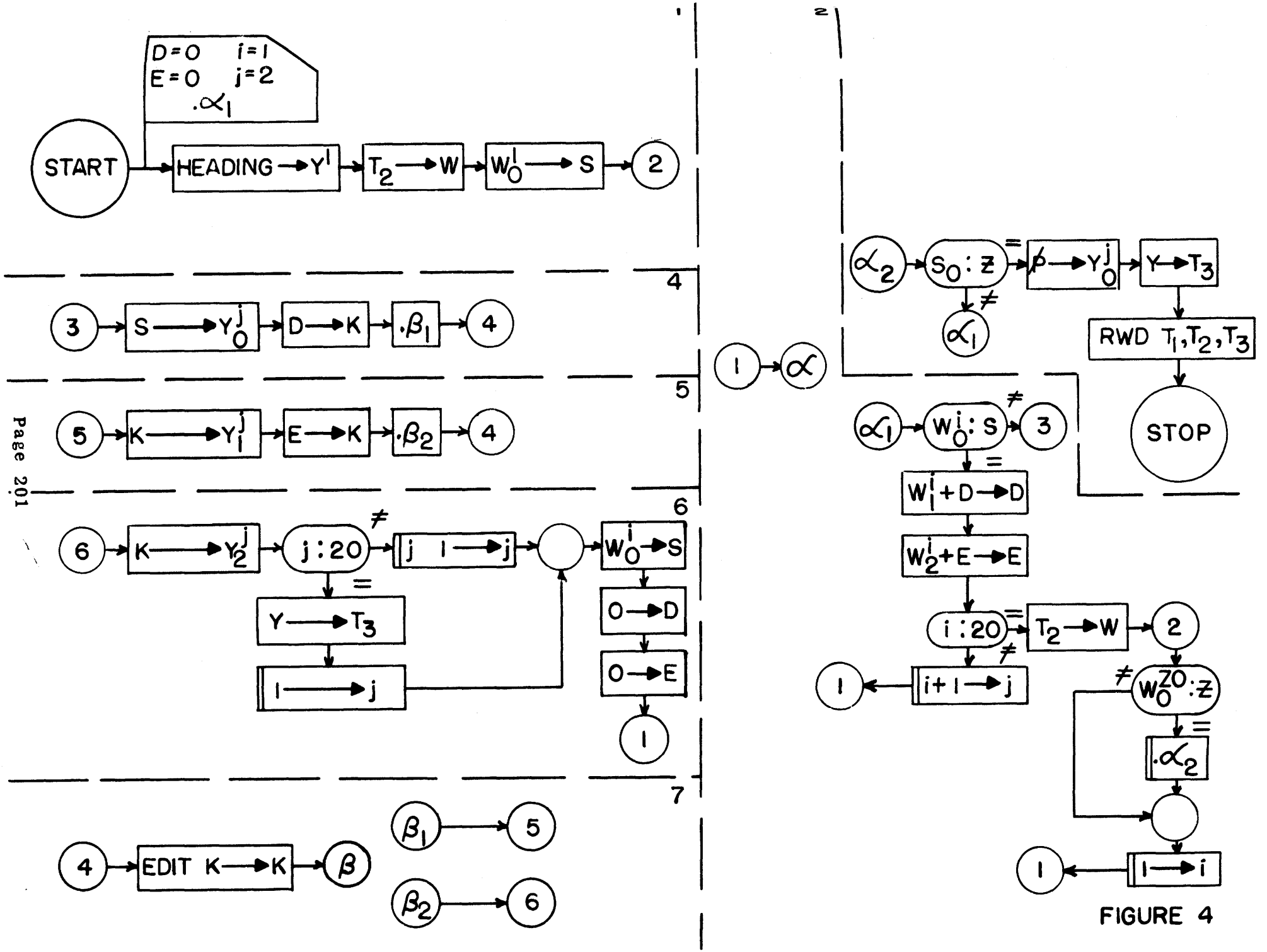

D=0 i=1
E=0 j=2
.α1
START
HEADING → Y^1
T2 → W
W0^1 → S
2
1
2
α2
S0 : Z
≠
=
β → Y0^j
Y → T3
RWD T1,T2,T3
STOP
α1
W0^i : S
3
W1^i + D → D
W2^i + E → E
i : 20
T2 → W
i + 1 → j
W0^20 : Z
.α2
1 → i
1 → α
4
3
S → Y0^j
D → K
.β1
4
5
5
K → Y1^j
E → K
.β2
4
6
6
K → Y2^j
j : 20
j 1 → j
W0^i → S
Y → T3
1 → j
0 → D
0 → E
1
7
4
EDIT K → K
β
β1 → 5
β2 → 6

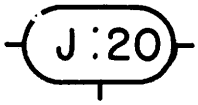
FIGURE 4

so that each and every item of the block is tested for a sentinel before it is processed. In either case the item counter i is reset to one and we return to the connector

Going back to the test

if the current item does not have a serial code identical with the serial code of the subtotals we are now accumulating, we know that these subtotals are now complete and we proceed to sections 4, 5, and 6, where the serial number and totals are edited and transferred into the output block. Using the variable connector

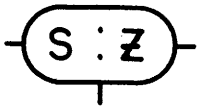
we can make the same editing routine serve to edit both D and E. The output item is labeled Y^j and when twenty such items have been accumulated, the equal-to path is taken from the test



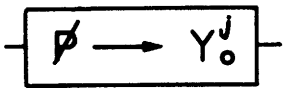
where the completed block is written on the blank tape mounted on UNISERVO 3, the output item counter j is reset to 1, S is set equal to the new serial number W_o^i, the subtotals D and E are reset to zero for the new accumulations, finally returning to connector

to process the new item.

Once a sentinel item has been located the equal-to path from the test



is taken.



Indicates that a printer stop is placed in the current output item, and the output block, whether complete or not, is then written onto the output tape. The tapes are then rewound and the computer stopped, terminating the problem.

Coding for Example #3

```
000    [11 000                    Second block of instructions -->
                32 060]                                    memory
001     B  056                    Set generalized overflow routine
                C  000
002     C  318                    Zero --> D
                C  319            Zero --> E
003     V  058
                W  260            Heading --> Y'
004     B  057
                C  262
005    32  200                    T2 --> W
                B  200            W'0 --> S
006     C  317
                U  038            Transfer control to connector 2
-------------------------------------------------------------------
007     L  317                    Variable connector α
                00 000                                        1
-------------------------------------------------------------------
008     B  200
                Q  026            Transfer control if W'0 = S    α
-------------------------------------------------------------------
009     L  318                    D --> K                     3
                J  320
010     R  053                    ·β1
                U  049            Transfer control to connector 4
-------------------------------------------------------------------
011     A  060
                C  318                                        5
012     L  319                    E --> K
                J  320
013     R  053                    ·β2
                U  049            Transfer control to connector 4
-------------------------------------------------------------------
014     A  061                                                6
                C  319
015     B  317
                L  318
```

```
016     [F45319
                  C  263]
017     [J  264
                  G  265]
018      B  016
                 *A  062

019      C  016                    j + 1 --> j
                  B  017
020      A  063
                  C  017

021      B  200                    W'0 --> S
                  C  317
022      C  318                    Zero --> D
                  C  319           Zero --> E
023      00 000
                  U  007           Transfer control to connector 1

024      73 260                    Y --> T3
                  V  064           1 --> j
025      W  016
                  U  021

026      B  318
                 A-  201           D + W'1 --> D
027      C  318
                  B  319
028     A-  207                    E + W'2 --> E
                  C  319
029      B  032
                  A  063
030      C  032                    i + 1  --> i
                  B  033
031     *A  066
                  C  033
```

```
032     [B  200
                    L  201]
033     [F42202
                    C  200]
034      J  201
                    G  202
035      00 000
                    U  007     Transfer control to connector  1
```

```
036     ZZZZZZ
                   ZZZZZZ
```

```
037     32 200                 T2 --> W
                   00 000
```

```
038     L  036                                                2
                    B  257
039     V  068
                    Q  041     Transfer control if W0^20 = Z
```

```
040     W  032                 1 --> i
                    U  007     Transfer control to connector 1
```

```
041     R  007                 · α2
                    U  040
```

```
042     L  317                                                α2
                    B  036
043     00 000
                    Q  045     Transfer control if S = Z
```

```
044     00 000
                    U  008     Transfer control if connector α1
```

```
045     B  067
                    81 000      Rewind instruction tape
046     R  017
                    U  016
-------------------------------------------------------

047     73 260                  Y --> T3
                    82 000      Rewind input tape
048     83 000                  Rewind output tape
                    90 000
-------------------------------------------------------

049     B  070                  1----- ------ --> rA
                    00 000
-------------------------------------------------------

050     .1 000
                    T  050
-------------------------------------------------------

051     C  321
                    F  321
052     B  071                  ΔΔΔΔΔ ΔΔΔΔΔ0 -->  2A
                    E  320
053     [000000                 Variable connector β
                    000000]
-------------------------------------------------------

054     B  322
                    A  072      Generalized overflow routine
055     C  322
                    U  322
-------------------------------------------------------

056     R00322
                    U00054
057     QUANTI
                    TYΔBRR
058     RSERIA
                    LΔCODE
059     TQUANT
                    ITYΔAT
```

```
060     000000
                  000000
061     000000
                  000000
062     003000
                  000003
063     000003
                  000003
064     F42319
                  C00260
065     J00261
                  G00262
066     003003
                  000000
067     ppppppp
                  ppppppp
068     B00200
                  L00201
069     F42202
                  C00200
070     1-----
                  ------
071     ΔΔΔΔΔΔ
                  ΔΔΔΔΔΔ
072     000000
                  000005
073     000000
                  000000
  .
  .
  .
  .
  .
119     000000
                  000000
```

SEC. 7. CONCLUDING REMARKS ON FLOW CHARTS

The development of logically consistent and clear flow-charts aid materially in detecting errors in the program, and pointing out labor and time saving shortcuts that may exist but have remained hidden because of the complexity of the problem. By requiring that the flow-chart not be a pictorial image of the coding the essential elements of the problem are laid clear for easy mastery by both the programmer and whoever is engaged in checking the coding. Obviously the development of a good notation for the elements of the problem eases the job of programming the particular problem, and conscientious effort to produce logically clear flow-charts leads to an early grasp of the techniques of efficient computer application. Many problems will require new symbology and these symbols should be concisely defined in a legend on the flow-chart.

So far, we have discussed flow-charts which made use of graphical equivalents of the three logical operations. Experience in preparing a wide variety of problems for solution on the UNIVAC has shown that larger building blocks exist that are common to many non-similar problems. That is, certain combinations of the Logical Operations, in themselves exceedingly complex, form a higher group of fundamental operations, and for many problems a grosser breakdown is first necessary. Collating or sorting routines, and various merging routines for instance are some of the basic building blocks of inventory control, billing, and tabulation problems. Diagrams involving these elements are often termed Process Flow-Charts. It is not the purpose of this chapter to discuss the preparation of Process Charts, but only to indicate that most large computer problems necessitate a preliminary breakdown in this fashion so that the essentials of the problem are cleared of the technical detail which a flow-chart of each step entails. This is quite permissable since the larger "fundamental" operations are well standardized routines.

Sec. 8. Notes on Multiple Input Routines

In most phases of complex commercial applications of UNIVAC, information from several different sources must undergo simultaneous processing. The information sources are recorded on tape, and each separate source is then assigned a UNISERVO which serves as the transport device enabling the information on the tape to be brought into the central computer in units of sixty words, or one block. As noted in the instruction code pertaining to the input orders, all information from the tapes must pass into the sixty-word register I, and thus, only a block of data from one UNISERVO may be read into the computer at a given time. There is an option present whereby the programmer may transfer the block of data present in rI into the memory for processing, and simultaneously order any particular UNISERVO to read another block of data into rI, this reading being done independent of the operation of the central computer which is free to begin calculations on the block transferred from rI.

Scrutiny of the previous sample problem will point out the fact that the block of data may not be completely processed when the UNISERVO has completed the transfer of the second block on tape into rI. Thus, the time to read a block of data from the tape may be completely absorbed in the computation time. This is true, of course, only if continuous read orders, the 3n or 4n, are given. For the combination 1n followed by 30, the lapse of time between the execution of the left instruction and the right instruction will be of the order of 100 milliseconds on the average. This is time during which the computer must wait until the read order is completed before it can execute the transfer from rI to the memory. Thus, from the standpoint of elapsed computer time it is desirable to do continuous read orders.

Where the processing to be done consists of bringing information into the computer from several different tapes and in an order which is not known in advance to the programmer, that is, in essentially a random fashion, how is the programmer to make sure that the data will be brought in at the right time and from the right tape?

For example, consider the problem basic to almost all commercial applications: Information, consisting of a series of items containing a serial number, are recorded on tape in ascending order by serial number. Two such sets of items (hereafter called A and B) are to be merged so as to produce one tape containing all the items present on both tapes, but arranged in ascending order.

The first block from each tape is brought into the computer, and their first items compared. The item with the smallest serial number is then transferred to the first position of an output block. If, for example, the lowest item in the comparison was from the A set of data, this item is placed in the output block. The next A item of the block is compared with this first B item and the smallest of these two items is sent to the second position of the output block. When the output block is filled, that is, when sufficient items have been transferred so that the output block contains 60 words, it is written on the output tape. The next lowest item transfer will go into the first output item position again. Soon one of the input blocks will be exhausted, all of its items having been transferred to the output block, therefore, we must bring in a new block of these items from tape. If we do the transfer from tape storage to computer by the sequence 1n -- 30, the computer must wait for approximately 100 milliseconds before it can execute the transfer from rI to the memory. But, if rI already contained the right block of data, we would be able to continue the processing by waiting only for the 3.5 millisecond transfer from register I to the memory. Since the order in which information from the two tapes will be read is indeterminate at the time of preparing the program, (the order of reading, of course, is determined by the data only) the programmer cannot always be sure that register I contains the proper block. There is, however, a method by which the programmer is assured that the information on tape will be processed in the proper order with the minimum of elapsed computer time. The method is known as the standby block procedure.

Initially the first two blocks of A items are brought into the computer and are known as block A_1 and A_2. The first block of B items is brought into the computer and is labeled B_1, but the second block is left in rI. The setup is shown in figure 1. The block for B_2 is shown shaded to indicate that it does not contain valid information.

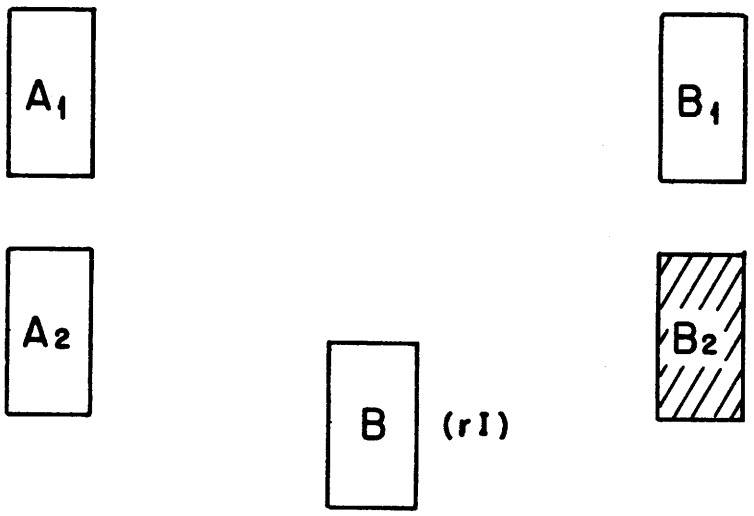


Figure 1

Items from the A_1 and B_1 blocks are then processed, and two possible conditions may obtain, either we exhaust all the A_1 items first, shown in figure 2, or all the B_1 items are exhausted first, shown in figure 3. Assume the condition shown in figure 2 is obtained.

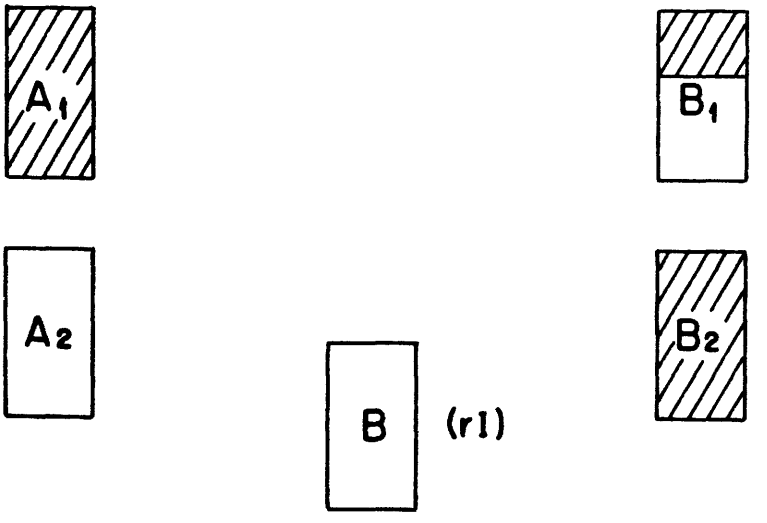


Figure 2

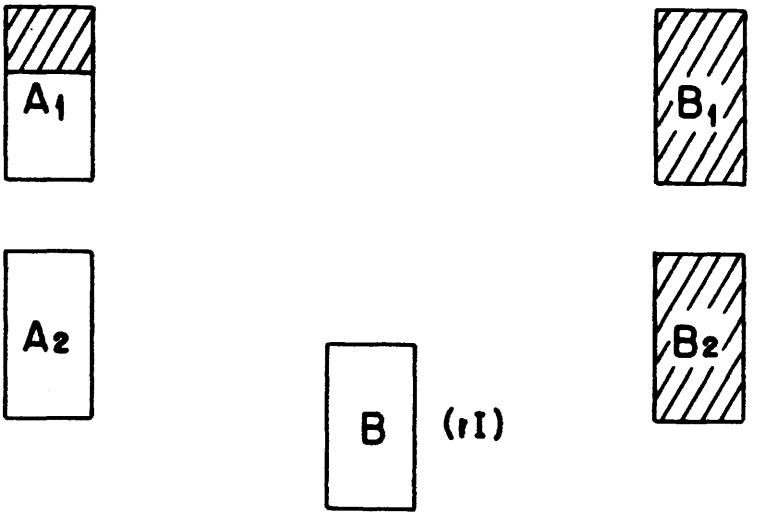


Figure 3

For this configuration we order the computer to transfer the block of B items in rI into the empty block B_2 and start filling rI with the next block of A items. While rI is being filled, we transfer the standby block of A items in A_2 into A_1, leaving the condition shown in figure 4. We return to the routine which continues the processing of A and B items from blocks A_1 and B_1 as before.

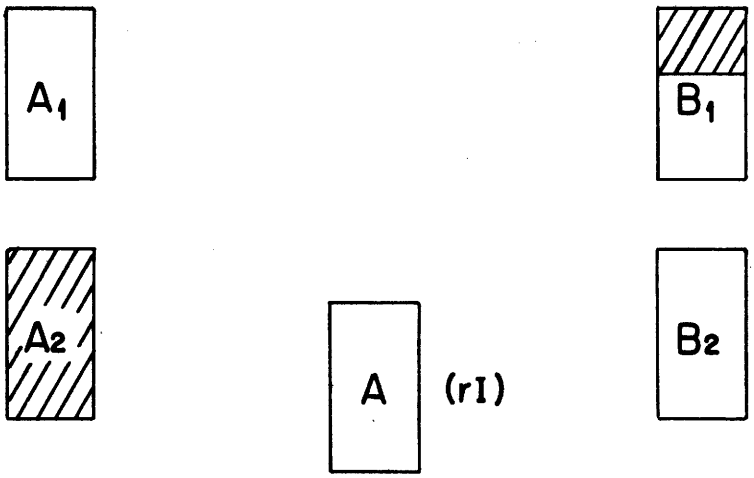


Figure 4

Now let us assume that the A_1 block is again exhausted before the rest of the items in B_1 are exhausted (figure 5).

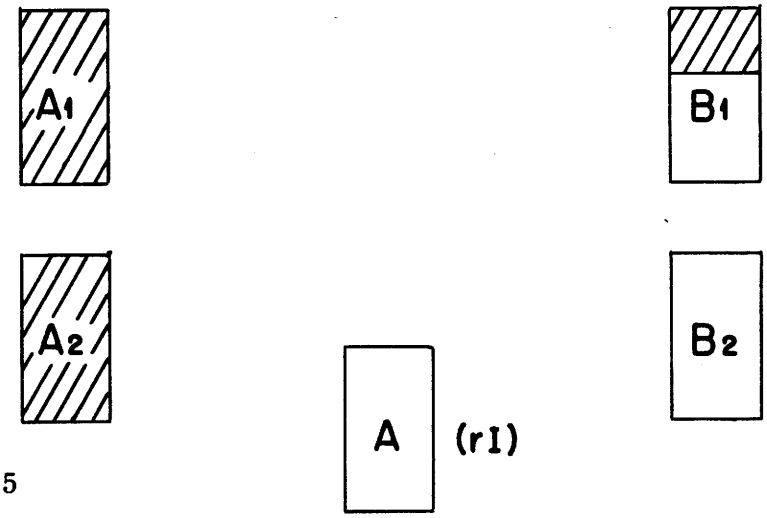


Figure 5

The contents of rI are transferred into the A_2 block which is now empty, and rI is filled from the next block on the A tape. Then the block A_2 is transferred into A_1 and we return to process the A_1 items against the remaining B_1 items (figure 6). If A_1 exhausts first again, we repeat the above sequence of operations, since the configuration is the same as figure 5.

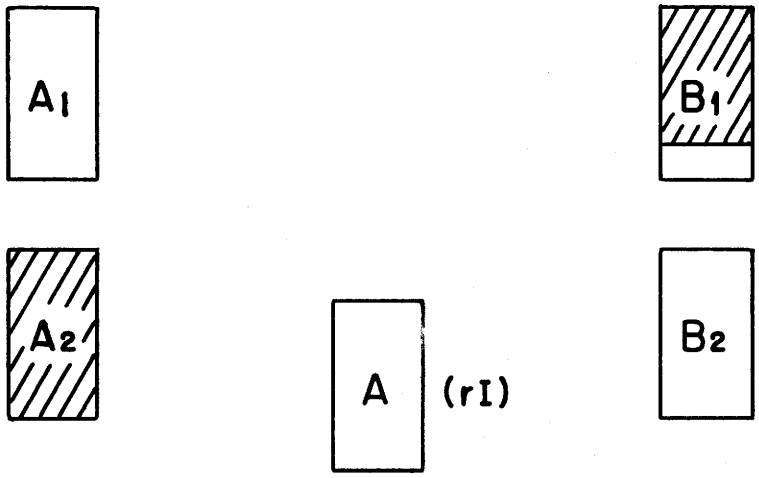


Figure 6

Let us now assume that the B_1 block is finally exhausted giving the condition shown in figure 7.

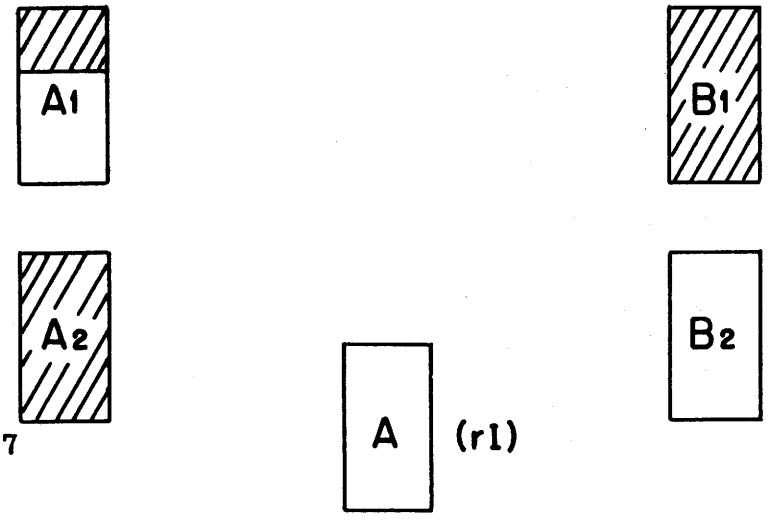


Figure 7

The A block in rI is transferred into the empty A_2 block, a block of B items is ordered from the B tape to be sent to rI, and then we transfer the B_2 block into B_1 and return to continue processing the remaining items in A_1 against the items now in B_1. The configuration is shown in figure 8.

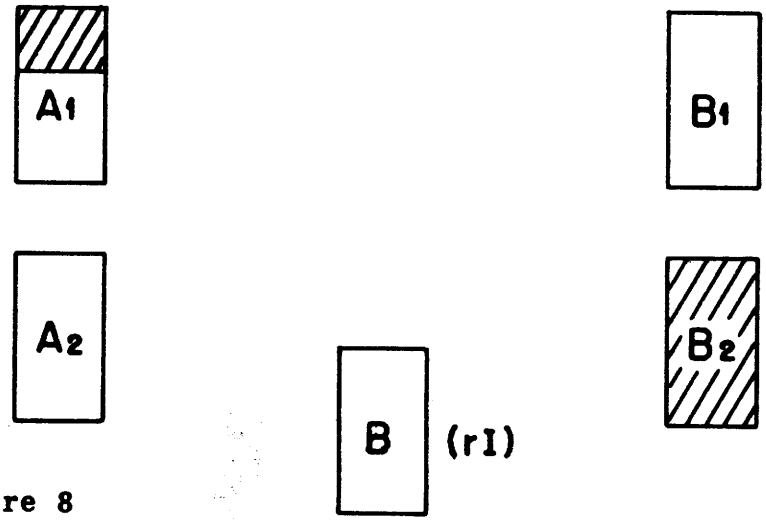


Figure 8

If now the A_1 block is exhausted first we have exactly the case shown in figure 2 and thus, repeat the sequence given above. If, however, the B_1 block is exhausted first we have the case shown in figure 3, not previously covered.

If then B_1 is exhausted first, we transfer the B block in rI into the empty B_2 block and order a B block from tape into rI. Meanwhile we transfer the just filled B_2 block into B_1, obtaining the pattern shown in figure 9, and then returning to the processing of these new B_1 items against the remaining A_1 items.

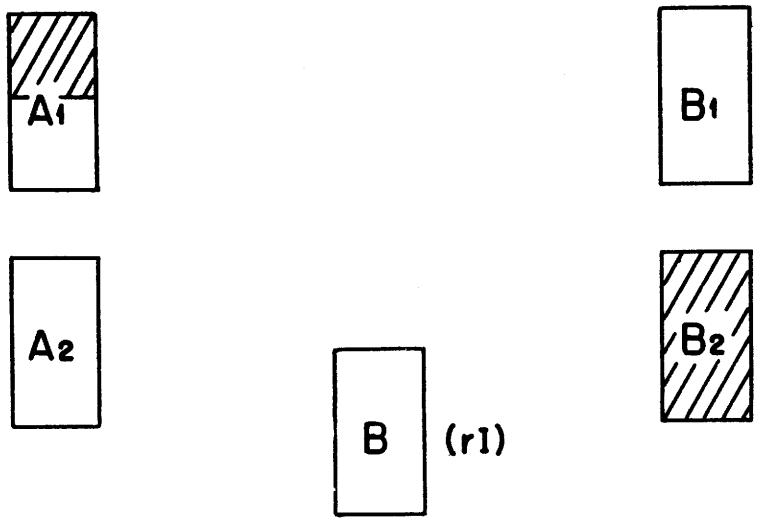


Figure 9

If A_1 now exhausts first we have the case of figure 2, repeated, while if B_2 exhausts first figure 3 is obtained.

Thus, we have covered all possible contingencies by this standard sequence of operations:

1) Transfer the block currently in register I into the proper standby block, A_2 if the items in rI are from tape A and into B_2 if the items are from Tape B. By remembering the last read order given we can always determine what items are in rI.

2) Order the next block of items from tape into rI. If an A block had been exhausted we order a block from tape A, but if a B block had been exhausted we order from tape B.

3) Transfer the standby block of items into the just exhausted block of items (A_2 --> A_1 if A_1 was emptied, B_2 --> B_1 if the B_1 block was exhausted) and return to the routine that continues the processing.

The coding necessary to carry out these steps is extremely simple. An example is given, where the A items are assumed to be on UNISERVO 2, the A_1 block occupies memory locations 760-819, A_2 being the block in 820-879. The B items are on UNISERVO 3, B_1 is the block from 880-939, and B_2 is the block from 940-999. At the start of the problem A_1, A_2, B_1 are filled, and rI contains the next B block.

Coding for Stand-by Block Example

010	00	000		
			Q 012	Transfer control if A_1 block is empty
011	00	000		
			U	Transfer control to continue processing
012	B	066		
			F 068	
013	R	064		
			U 062	Transfer control to read routine
014	Y	820		
			Z 760	
015	Y	830		
			Z 770	
016	Y	840		
			Z 780	A_2 --> A_1
017	Y	850		
			Z 790	
018	Y	860		
			Z 800	
019	Y	870		
			Z 810	
020	00	000		
			U	Transfer control to reset A item counter
031	00	000		
			Q 033	Transfer control if B_2 block is empty
032	00	000		
			U	Transfer control to continue processing
033	B	067		
			F 068	

034	R 064		
		U 062	Transfer control to read routine
035	Y 940		
		Z 880	
036	Y 950		
		Z 890	
037	Y 960		
		Z 900	B_2 --> B_1
038	Y 970		
		Z 910	
039	Y 980		
		Z 920	
040	Y 990		
		Z 930	
041	00 000		
		U	Transfer control to reset B item counter
062	E 065		
		C 063	
063	[000000		Read routine
		000000]	
064	[000000		
		000000]	
065	[J00065		
		330940]	
066	J00065		
		320820	
067	J00065		
		330940	
068	000000		
		000111	

Chapter II

Appendix

Sec. I. Review Practice Exercises

1. In memory locations 100...129 are 30 randomly arranged numbers. Among these 30 numbers one and only ine is negative. Code a routine, starting in memory location 010, to find the single negative number and print it out on the typewriter for inspection.

2. Two sets of numbers are stored in the memory, the first set (A's) in 100...199, the second set (B's) in 200...299. Code a routine that will perform the operation $A_i + B_i \rightarrow C_i$, i going from 0 to 99. Let the sum's (C's) be stored in 300...399.

3. In memory locations 100...199 are contained a table of 100 values for a function corresponding the arguments .000, .001, .002, ..., .099. In memory location 200 is an argument within the range of the table. Using linear interpolation, find the value of the function which corresponds to the argument in M.L. 200. Then print out the argument and interpolated function value.

4. Given an integer X, $0 < X \leq 9$, stored in memory location 200. Compute X! and print both X and X! on typewriter.

5. Stored in 13 consecutive memory locations starting with memory location 100 are coded values for 13 playing cards. Determine if the 13 cards contain a royal flush (Ace, King, Queen, Jack, and Ten) in spades or hearts. Each word contains the value of a card.

```
000 00S 000 0CC

    1 = Clubs                02 = Duce
                             03 = Trey
    2 = Diamonds             '     '
S                        CC  '     '
    3 = Hearts               '     '
                             12 = Queen
    4 = Spades               13 = King
                             14 = Ace
```

6. Code a routine which will compute all values for the polynomial function: $y = 4X^4 + 3X^2 - \frac{X}{5} + 1$, in the range $.01 \leq X < 1.00$, with $\Delta X = .001$. Maintain nine significant digits in the computed values. Write the values on tape as two word items, the argument in the first word and the value of the function in the second word.

7. Tape unit #2 contains 10 word items (A's). These items are arranged in ascending order by a key which is the first word of each item. If the last block is not filled with A items, it will contain sufficient sentinel items (ZZZ's in the first word of each item) to fill the block. Two additional sentinel blocks (ZZZ's in words 000 and 050) will follow the partial block. If there is no partial block, only the two sentinel blocks will follow the last A block.

Tape unit #3 contains B items similar to the A items, with the same ending sentinel convention.

Code a routine that will merge the items of both tapes together to form a continuous string of items in ascending order, with the same ending sentinel convention as the input tapes. You may have more than one tape of output (2000 blocks). If so, the first output tape should contain no partial blocks. Assume you have five tape units for the problem.

8. Ten numbers are stored in memory locations 100...109. Prepare a routine that will arrange these numbers in ascending numerical order.

9. Code a routine to find the value of the integral:

$$y = \int_1^{\infty} \frac{1}{X^3}\, dx.$$

NOTE:

$$\int_1^{\infty} \frac{1}{X^3}\, dx = \sum_{1}^{10,000} \frac{1}{X^3}\, \Delta X = .01 \left[\frac{1}{(1)^3} + \frac{1}{(1.01)^3} + \frac{1}{(1.02)^3} + \ldots \right]$$

10. Tape unit number two contains a variable number of 10-word items. Each item contains the following information:

WORD NO.			
0	SSSSSS	SSSSSS	Stock Number
1	SSSSSS	SSSSSS	
2	00000X	XXXXXX	Total Required
3	------	------	Other information
4	------	------	
5	00000X	XXXXXX	On hand quantity
6	XXXXXX	000000	Date
7	00000X	XXXXXX	On order quantity
8	------	------	Other information
9	------	------	

This tape contains the standard ending sentinel convention described in problem number seven. Code a routine that will select and write on tape unit number three all the items of stock number:

AM4367 820000

V63900 W98800,

that have a total required greater than the sum of the on hand and on order quantities.

11. There are sixty open policies (premiums not paid) in the files of a fire insurance company. For this problem, assume that information concerning the policies, when put in one-word storage, would take the form

```
Policy
Number Type Premium

  XXXX  XXX  XXXXX
```

This data is in random order on the second block of the tape on UNISERVO 1. It is required to set up a programming routine which will arrange this data in the order of policy numbers. Print these sixty words, in the order of policy numbers, on Supervisory Control and write the arrangement on the tape on UNISERVO 2. Also, total the unpaid premiums, and type on Supervisory Control. Make any other assumptions necessary to the solution of this problem.

Note: Section 2 in the Appendix describes two methods of sorting.

12. Set up a routine which will evaluate.

$$y = 13.1X^2 - 2.72X - 8.44$$

to four significant places for values of X from 0.01 to 1.00 in intervals of 0.01. Arrange to have the result typed on Supervisory Control and also put on tape to be used on the UNIPRINTER. The results are to be edited in two columns, properly labeled, one for the arguments and the other for the computed results.

13. Consider the miniature production problem tabulated below.

		Time Periods					(5)
		1	2	3	4	5	Totals
A-Autos				5	10	4	19
(1) B-Wheels			25	50	20		95
C-Steel	(2) Autos		2.75	5.5	2.2		10.45
	(3) Wheels	0.25	0.50	0.2			0.95
	(4) Total	0.25	3.25	5.7	2.2		11.4

a. 5, 10 and 4 automobiles are required in the three time periods shown.

b. Five wheels are needed for each auto, one time period in advance.

c. Each auto requires .6 ton of steel, including wheels, one time period in advance.

d. Each wheel requires .01 ton of steel in advance.

Beginning with the automobile requirements, determine a routine which will provide, in each time period,

1. The number of wheels.

2. The total tons of steel.

and the overall totals for each item.

In other words, it is required to set up a routine to give the results shown in rows 1, 2, 3, 4 and column 5, of the table given above.

SEC. 2. TECHNIQUES OF SORTING, ARRANGING, ORDERING

Example 1:

Consider the numbers

2 3 1 6 5 4

to be arranged in numerical order.

1. Compare

   the first number with the second, take smaller and compare with the third, again take the smaller and compare with the fourth etc. This will isolate the smallest number, 1, which is put in storage and replaced by a Z in the sequence

2. This gives us

						in storage
2	3	Z	6	5	4	1

Start again with the first number and repeat the comparisons.

3. We now have

						in storage	
Z	3	Z	6	5	4	1	2

4. This repeated until complete arrangement is obtained

*Note that Z will show up greater than any numeric digit.

Example 2:

To arrange in numeric order

32, 53, 72, 18, 41, 75, 48, 23, 27, 61, 11, 6, 4, 40

1. Apply the equality tests on the units digits to give:

Unit's Digit

0	1	2	3	4	5	6	7	8	9
40	41	32	53	04	75	06	27	18	
	61	72	23					48	
	11								

2. Then sort by ten's digits starting on the left to give:

Ten's Digit

0	1	2	3	4	5	6	7	8	9
04	11	23	32	40	53	61	72		
06	18	27		41			75		
				48					

(and then by hundreds digits, if necessary.)

SEC. 3. UNIVAC INSTRUCTION CODE C-10

The C-10 Code is compiled and restated here for the convenience of the reader. Although much of the information provided herein has already been defined some of the material is new. Specifically, the average times in microseconds for each of the instructions is not given in the body of the manual but is included in this section. Also, certain combinations of rewind instructions will cause UNIVAC to stall; hence, a short discussion of these possibilities is provided.

UNIVAC INSTRUCTION CODE C-10

Av. Time in Microseconds	Instruction	Description
525	Am	Add (m) to (rA), result in rA; (m) also placed in rX.
445	Bm	Clear (rA), then bring (m) into rA; (m) also placed in rX.
445	Cm	Place (rA) in m. Clear rA.
3890	Dm	Divide (m) by (rL) rounding off the quotient to 11 digits; result in rA. Unrounded quotient in rX.
445	Em	Extract from (m) the characters (including digits) specified by (rF). Clear only those characters of rA which are replaced by the extracted characters. When a digit in rF is "0", leave the corresponding character in rA unaltered. When a digit in rF is "1", insert the corresponding character of m in rA.
445	Fm	Place (m) in rF.

445	Gm	Place (rF) in m.
445	Hm	Place (rA) in m without clearing rA. (i.e., Hold (rA) in rA).
445	Jm	Place (rX) in m; (rX) unaltered.
285	Km	Place (rA) in rL, clear rA; disregard m.
445	Lm	Place (m) in rL; (m) also placed in rX.
2150	Mm	Multiply (rL) by (m) rounding off the product to 11 digits; result in rA.
2150	Nm	Negative Multiplication. Multiply (rL) by -(m) rounding off the product to 11 digits; result in rA.
2150	Pm	Multiply (rL) by (m), storing the more significant half of the product in rA and the less significant half in rX.
365	Qm	Equality Test. Transfer control to m if (rA) = (rL).
445	Rm	Record the number of the control count plus one as an unconditional transfer instruction in m. U(c+1) in m.
525	Sm	Subtract (m) from (rA), result in rA; -(m) also left in rX.
365	Tm	Test to determine whether (rA) > (rL) algebraically; if so, transfer control to m.

285	Um	Unconditional transfer of control to m.
485	Vm	Place 2 consecutive words, starting with m, in rV; m should be a multiple of 2. (For exceptions see additional information in Chapter 5).
485	Wm	Place (rV) in 2 consecutive memory locations starting with m; m should be a multiple of 2. (For exceptions see additional information in Chapter 3).
285	Xm	Add (rX) to (rA), result in rA, disregard m. (rX) unaltered.
650	Ym	Place 10 consecutive words, starting with m, in rY. Here m must be an integral multiple of 10.
650	Zm	Place (rY) in 10 consecutive memory locations, starting with m. Here m must be an integral multiple of 10.
285	OOm	Pass to next instruction (Skip instruction).

SHIFT INSTRUCTIONS. N RANGES FROM I TO 9

40.5n+245	.nm	Shift all digits of rA, including the sign position, n digits to the right dropping the n right-hand digits. Disregard m.
40.5n+245	;nm	Shift all digits of rA, including the sign position, n digits to the left, dropping the n left-hand digits. Disregard m.

40.5n+245	-nm	Shift all digits of rA, except the sign position, n digits to the right, dropping the n right-hand digits (equivalent to multiplying (rA) by 10^{-N} without rounding). Disregard m.
40.5n+245	0nm	Shift all digits of rA, except the sign position, n digits to the left, (equivalent to multiplying (rA) by 10^{N} without rounding). Disregard m.

TAPE INSTRUCTIONS FOR 1 TO 9 TAPES

3500*	1nm	Read one block of data (60 words) from tape n and store in rI, tape moving in a forward direction; disregard m.
3500*	2nm	Read one block of data (60 words) from tape n and store in rI, tape moving in a backward direction; disregard m.
3500*	3nm	Transfer data (60 words) previously stored in rI to 60 consecutive memory locations, beginning with m, where m is an integral multiple of 10; then read one block of data (60 words) from tape n and store in rI, tape moving in a forward direction.
3500*	4nm	Transfer data (60 words) previously stored in rI to 60 consecutive memory locations, beginning with m, where m is an integral multiple of 10; then read one block (60 words) from tape n and store in rI, tape moving in a backward direction.

3500*	5nm	Write 60 consecutive words, starting with m, where m is an integral multiple of 10, on tape n; moving in a forward direction. Pulse density 100/in. (tape to be used in a future UNIVAC operation.)
3500*	6nm	Rewind tape n to the beginning. Disregard m.
3500*	7nm	Write 60 consecutive words, starting with m, where m is an integral multiple of 10, on tape n; tape moving in a forward direction. Pulse density 20/in. (Tape to be used in future UNIPRINTER or UNIVAC operations.)
3500*	8nm	Rewind tape n to beginning and set an interlock, disregard m. The setting of the interlock will produce a visual signal and no data can be read from or written on tape n until the mechanical interlock release switch on the UNISERVO has been actuated. More than one tape may be rewound simultaneously.

TAPE INSTRUCTIONS FOR TAPE NUMBER 10

Instructions for tape 10 operate in the same manner as those for tapes 1 to 9 except that n is not a number but a "minus sign."

Supervisory Control Instructions with Input and Output Selector Switch set at Position 1.

3500*	10m	Stop UNIVAC operations and produce a visual signal. Call for one word to be typed from the Supervisory Control keyboard into m of UNIVAC. UNIVAC operations are resumed after the Word Release Button on the Supervisory Control has been actuated.
3500*	50m	Print (m), one word, on Printer associated with Supervisory Control. UNIVAC operations are resumed automatically after (m) has been transferred to an intermediate output storage location prior to printing.

Supervisory Control Output Switch - operates in conjunction with the 50 m instruction.

Position 1 Normal. The 50 m instruction operates as described above.

285

Position 2 Skip. The 50 m instruction is decoded as a skip instruction (00m).

Position 3 Stop. Stop UNIVAC operations and produce a visual signal. At this time the setting of the Supervisory Control Output Selector switch may be changed. Actuation of the start button causes the 50 m instruction to be performed.

*Times listed for Tape and Supervisory Control operations represent computer time only. Computations may proceed after the lapse of the indicated times, but may be interrupted if another tape instruction of the same nature (read followed by read, or write followed by write) occurs before the first tape reading or writing operation has been completed. The time to read or write one block of information is approximately .085 seconds for 100/ in pulse density if no tape reversal is needed. If tape is reversed an additional 0.6 second is required. The first block to be read from or written on a tape required approximately 1.8 seconds. The Printer on Supervisory Control operates at approximately 10 characters per second.

STOP INSTRUCTIONS

285	,m	Breakpoint Stop, used for checking of programming; disregard m. If the two-position Breakpoint switch on the Supervisory Control is in the "Normal" position, interpret as a skip instruction. If the switch is in the "Breakpoint" position, interpret as a stop instruction. To resume UNIVAC operations actuate Start Button.
	Qnm or Tnm	Conditional Transfer Breakpoint Stop, used for checking of programming. The Breakpoint Stop operates in conjunction with the setting of 12 conditional Transfer Breakpoint Selector Buttons on Supervisory Control; "Reset", "0...9", "All". Reset - Qm and Tm instructions operate in the normal manner.

0...9 - If n corresponds to the setting of the button depressed, the UNIVAC stops, after (rA) and (rL) have been compared but before the transfer takes place. One or more of the Buttons labeled 0...9 may be depressed simultaneously.

All - The UNIVAC stops after (rA) and (rL) have been compared on all Qm and Tm instructions. To complete instruction actuate Start Button.

285 90m Stop UNIVAC operations and produce a signal; disregard m.

Rewind Instructions where "n" in each case is the same

(a) The following combinations will <u>not</u> cause UNIVAC to stall

6n followed by 8n
6n followed by 6n
tape "n" not used followed by 8n
tape "n" not used followed by 6n

(b) The following combinations will cause UNIVAC to stall

8n followed by 8n
8n no tape on Servo n
6n no tape on Servo n
8n followed by any other tape order on Servo n prior to substitution of new tape.

SEC. 4. TABLES SHOWING COMPUTER RESPONSES TO CERTAIN SPECIAL CONDITIONS

Table 1 shows the effect of the "S" order on the sign column and is an extension of the discussion in Section 11 of Chapter 4. Referring to the table, typewriter characters are arranged in groups of two rows with four symbols in each row. The six-pulse representation of each character is also shown. The table is read as follows: An "S" instruction on any of the four symbols in the top row or a group will produce the symbol directly below in the second row of the group. For example, to determine the effect of an "S" order the symbol *í*, with a pulse representation of 00 000, this symbol is first located in the top row of the first group of those on the left. The symbol resulting from this instruction is Δ found in the second row of this group directly under the *í*. Similarly, the result of acting on D, in the sign column, leads to the letter C; acting on C leads to D; 7 leads to 8 and 8 leads to 7.

Table 2 shows the results of multiplying digits and other typewriter characters. To use the table the multiplicand is found in the sets of characters at the top and the multiplier is found on the left. The number (or characters) in the body of the table located in the space corresponding to the column containing the multiplicand and row containing the multiplier is the product produced by the computer. For example, any one of the quantities 5, E, N or J multiplied by anyone of the quantities 2, B, K or S will produce 10. This will be clear when it is recalled that the zone indicators are ignored in the multiplication process.

TABLE 1 EFFECT OF S ORDER ON SIGN COLUMN

	00	01	10	11
0000 0001	/ Δ	/ ,	/ \	/ B
0010 0011	– 0	· ;	X +	\ \
0100 0101	1 2	A B	J K	/ S
0110 0111	3 4	C D	L M	T U
1000 1001	5 6	E F	N O	V W
1010 1011	7 8	G H	P Q	X Y
1100 1101	9 \	I \	R /	Z Ø
1110 1111	\ \	\ \	\ Ø	\ \

	00	01	10	11
0001 0000	Δ /	, /	\ /	B /
0011 0010	0 –	; ·	+ X	\ \
0101 0100	2 1	B A	K J	S /
0111 0110	4 3	D C	M L	U T
1001 1000	6 5	F E	O N	W V
1011 1010	8 7	H G	Q P	Y X
1101 1100	\ 9	\ I	/ R	Ø Z
1111 1110	\ \	\ \	Ø \	\ \

/ NOT PRESENTLY DEFINED

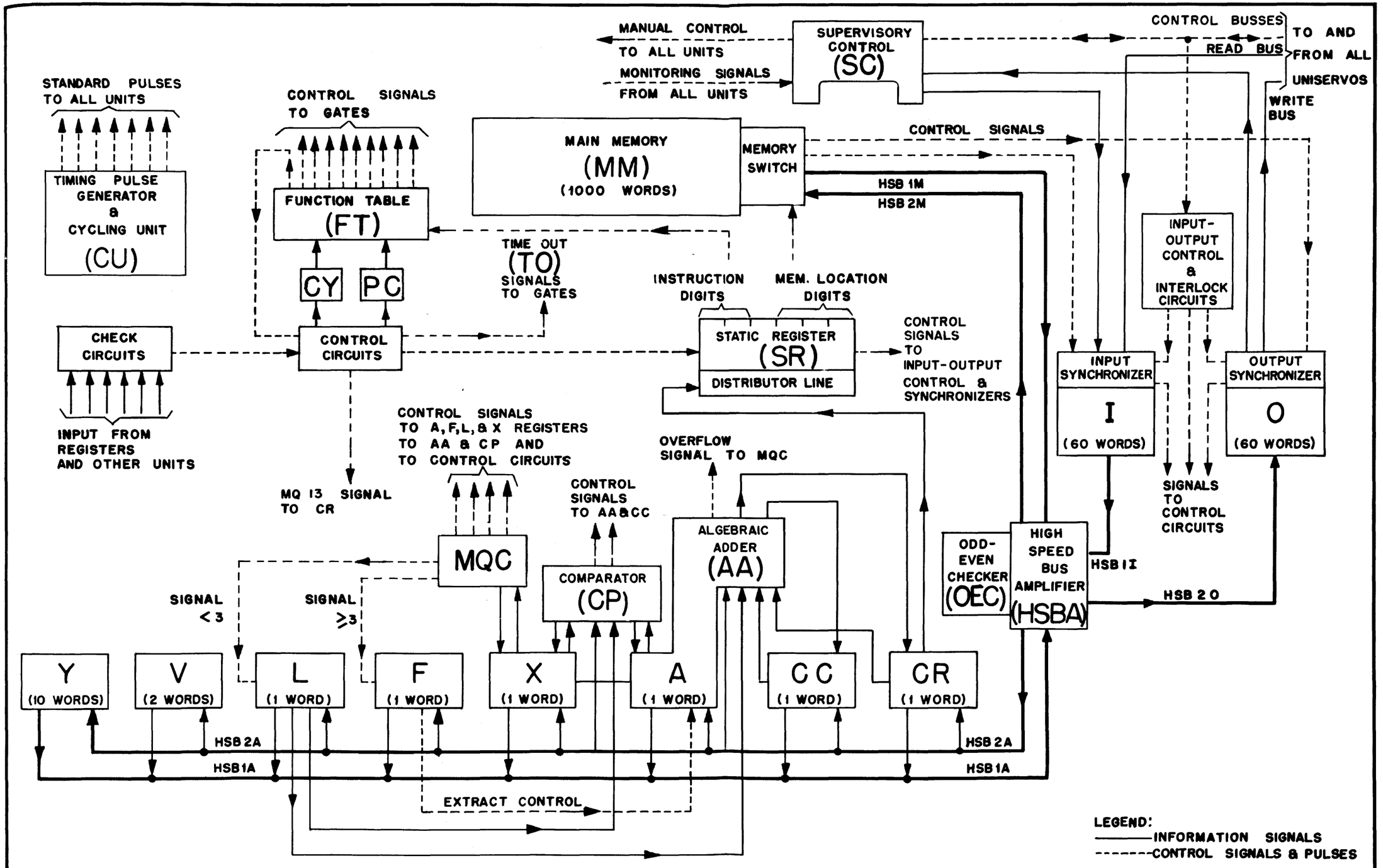
X NOT AVAILABLE (USED INTERNALLY)

TABLE 2 MULTIPLICATION OF TYPEWRITER CHARACTERS

MULTI-PLIER	MULTIPLICAND															
	/̸ R̸ 7̸ P̸	Δ , \ B̸	- · X \	0 ; / \	1 A J +	2 B K S	3 C L T	4 D M U	5 E N V	6 F O W	7 G P X	8 H Q Y	9 I R Z	\ \ L̸ Ø	\ \ \ \	\ \ U̸ \
0 ; / \	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1 A J +	/̸	Δ	–	0	1	2	3	4	5	6	7	8	9	10	11	12
2 B K S	α	β	Δ	0	2	4	6	8	10	12	14	16	18	20	22	24
3 C L T	7	10	/̸	0	3	6	9	12	15	18	21	24	27	30	33	36
4 D M U	4	1Δ	β	0	4	8	12	16	20	24	28	32	36	40	44	48
5 E N V	1	1β	11	0	5	10	15	20	25	30	35	40	45	50	55	60
6 F O W	14	20	10	0	6	12	18	24	30	36	42	48	54	60	66	72
7 G P X	11	2Δ	1–	0	7	14	21	28	35	42	49	56	63	70	77	84
8 H Q Y	1Δ	2β	1Δ	0	8	16	24	32	40	48	56	64	72	80	88	96
9 I R Z	21	30	1/̸	0	9	18	27	36	45	54	63	72	81	90	99	108
\ \ L̸ Ø	2Δ	3Δ	1β	0	10	20	30	40	50	60	70	80	90	100	110	120
\ \ \ \	2γ	3β	21	0	11	22	33	44	55	66	77	88	99	110	121	132
\ \ U̸ \	28	40	20	0	12	24	36	48	60	72	84	96	108	120	132	144
/̸ R̸ 7̸ P̸	25	4Δ	2–	0	13	26	39	52	65	78	91	104	117	130	143	156
Δ , \ B̸	22	4β	2Δ	0	14	28	42	56	70	84	98	112	126	140	154	168
– · X \	2–	50	2/̸	0	15	30	45	60	75	90	105	120	135	150	165	180

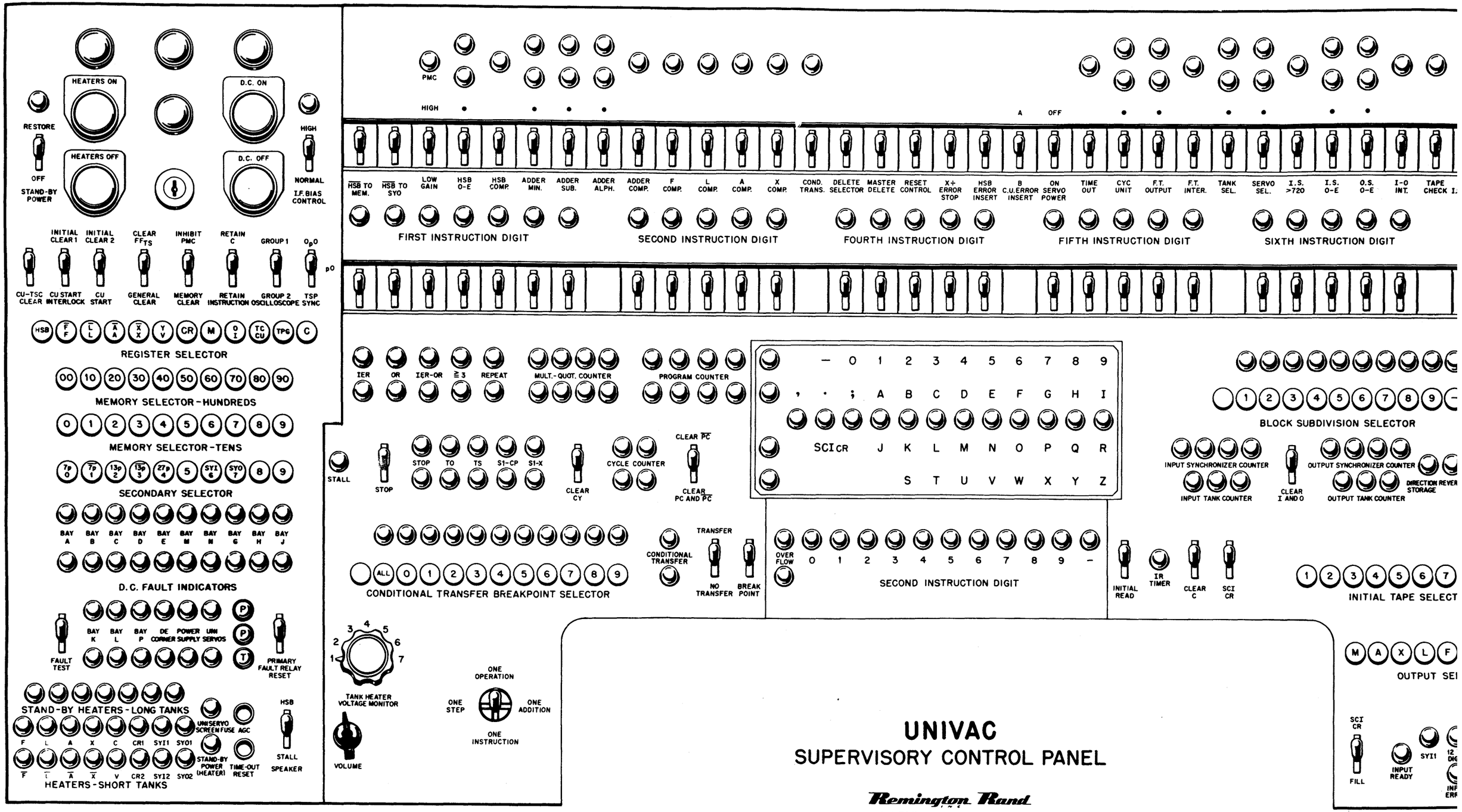
/ NOT PRESENTLY DEFINED
X NOT AVAILABLE (USED INTERNALLY)
α = 1101
β = 1111

γ = 1110
1Δ = 0100 0001
2/̸ = 0101 0000
3β = 0110 1111



ECKERT-MAUCHLY DIVISION OF REMINGTON RAND INC.

SIMPLIFIED BLOCK DIAGRAM OF THE UNIVAC EBU-100


HEATERS ON
HEATERS OFF
D.C. ON
D.C. OFF
RESTORE
OFF
STAND-BY POWER
HIGH
NORMAL
I.F. BIAS CONTROL
INITIAL CLEAR 1
INITIAL CLEAR 2
CLEAR FF TS
INHIBIT PMC
RETAIN C
GROUP 1
O p O
p O
CU-TSC CLEAR
CU START INTERLOCK
CU START
GENERAL CLEAR
MEMORY CLEAR
RETAIN INSTRUCTION
GROUP 2 OSCILLOSCOPE
TSP SYNC
REGISTER SELECTOR
MEMORY SELECTOR - HUNDREDS
MEMORY SELECTOR - TENS
SECONDARY SELECTOR
D.C. FAULT INDICATORS
FAULT TEST
PRIMARY FAULT RELAY RESET
STAND-BY HEATERS - LONG TANKS
HEATERS - SHORT TANKS
UNISERVO SCREEN FUSE
AGC
STAND-BY POWER (HEATER)
TIME-OUT RESET
HSB
STALL
SPEAKER
PMC
HIGH
HSB TO MEM.
HSB TO SYO
LOW GAIN
HSB O-E
HSB COMP.
ADDER MIN.
ADDER SUB.
ADDER ALPH.
ADDER COMP.
F COMP.
L COMP.
A COMP.
X COMP.
COND. TRANS.
DELETE SELECTOR
MASTER DELETE
RESET CONTROL
X+ ERROR STOP
HSB ERROR INSERT
B C.U. ERROR INSERT
ON SERVO POWER
TIME OUT
CYC UNIT
F.T. OUTPUT
F.T. INTER.
TANK SEL.
SERVO SEL.
I.S. >720
I.S. O-E
O.S. O-E
I-O INT.
TAPE CHECK I.
FIRST INSTRUCTION DIGIT
SECOND INSTRUCTION DIGIT
FOURTH INSTRUCTION DIGIT
FIFTH INSTRUCTION DIGIT
SIXTH INSTRUCTION DIGIT
IER
OR
IER-OR
≧ 3
REPEAT
MULT.-QUOT. COUNTER
PROGRAM COUNTER
STALL
STOP
STOP
TO
TS
S1-CP
S1-X
CLEAR CY
CYCLE COUNTER
CLEAR PC
CLEAR PC AND PC
CONDITIONAL TRANSFER BREAKPOINT SELECTOR
CONDITIONAL TRANSFER
TRANSFER
NO TRANSFER
BREAK POINT
SECOND INSTRUCTION DIGIT
INITIAL READ
IR TIMER
CLEAR C
SCI CR
BLOCK SUBDIVISION SELECTOR
INPUT SYNCHRONIZER COUNTER
INPUT TANK COUNTER
CLEAR I AND O
OUTPUT SYNCHRONIZER COUNTER
OUTPUT TANK COUNTER
INITIAL TAPE SELECT
OUTPUT SEL
TANK HEATER VOLTAGE MONITOR
VOLUME
ONE OPERATION
ONE STEP
ONE ADDITION
ONE INSTRUCTION
SCI CR
FILL
INPUT READY
SYI1
UNIVAC
SUPERVISORY CONTROL PANEL
Remington Rand
ECKERT-MAUCHLY DIVISION

PMC

HIGH • • • • A OFF • • • • • • S

LOW GAIN | HSB O-E | HSB COMP. | ADDER MIN. | ADDER SUB. | ADDER ALPH. | ADDER COMP. | F COMP. | L COMP. | A COMP. | X COMP. | COND. TRANS. | DELETE SELECTOR | MASTER DELETE | RESET CONTROL | X÷ ERROR STOP | HSB ERROR INSERT | B C.U.ERROR INSERT | ON SERVO POWER | TIME OUT | CYC UNIT | F.T. OUTPUT | F.T. INTER. | TANK SEL. | SERVO SEL. | I.S. >720 | I.S. O-E | O.S. O-E | I-O INT. | TAPE CHECK | R I.S.ERROR INSERT | I.S. ERROR CLEAR | O.S. ERROR CLEAR

RST INSTRUCTION DIGIT — SECOND INSTRUCTION DIGIT — FOURTH INSTRUCTION DIGIT — FIFTH INSTRUCTION DIGIT — SIXTH INSTRUCTION DIGIT

←DECIMAL ZERO→

←BINARY ZERO→ INST. MSR SR

ABSOLUTE REF. VOLTAGE

LOW -100 -50 -40 -30 -20 0 20 40 50 HIGH 100 — PERCENT OF ALLOWABLE DEVIATION — 100 PER CENT

TANK HEATER VOLTAGE

VOLTAGE MON REMOTE METER

0	1	2		3	4	5
+750 / 0	+90 / 0	+8 / -74	0	-4 / -74	-88 / -117	+290 / +90
+410 / +280	+80 / +70	+5 / 0	1	-14 / -34	-88 / -150	+280 / +246
+380 / +90	+80 / +60	+5 / -14	2	-20 / -40	-95 / -100	+246 / +165
+375 / 0	+80 / 0	0 / -175	3	-20 / -100	-95 / -117	+120 / +90
+280 / 0	+80 / -106	0 / -11	4	-25 / -100	-100 / -175	0 / -34
+246 / 0	+75 / 0	0 / -13	5	-25 / -190	-100 / -216	-34 / -74
+200 / 0	+70 / +60	0 / -14	6	-34 / -117	-88 / -100	
+165 / +90	+70 / 0	0 / -15	7	-34 / -50	-117 / -190	0 / -191S
+165 / +60	+66 / +60	0 / -16	8	-40 / -55	-125 / -190	-166S / -191S
+160 / -40	+60 / 0	0 / -17	9	-40 / -100	-140 / -190	-191S / -217S
+150 / 0	+50 / 0	0 / -18	10	-50 / -150	-150 / -163	+48 / 0
+120 / +105	+40 / 0	0 / -20	11	-50 / -190	-150 / -165	
+120 / -30	+40 / -88	0 / -21	12	-55 / -100	-150 / -190	
+105 / +95	+35 / 0	0 / -40	13	-55 / -150	-150 / -216	
+105 / +80	+30 / 0	0 / -50	14	-60 / -88	-165 / -216	
+95 / 0	+30 / -40	0 / -60	15	-60 / -100	-175 / -216	+279 / +90
+90 / +84	+20 / 0	0 / -100	16	-71 / -88	-190 / -203	+247 / +90
+90 / +80	+15 / -100	0 / -106	17	-74 / -88	-190 / -300	+79 / 0
+90 / +70	+8 / 0	0 / -300	18	-74 / -190	-216 / -229	-74 / -86
+90 / +60	+8 / -14	-4 / -34	19	-80 / -150	-216 / -300	-100 / -150

MEASURE

DELAYED SHUT-OFF

VOLTAGE MON. MANUAL SWITCHING

HIGH

VOLTAGE MON. FAULT RELAY RESET

LOW

IER-OR ≧3 REPEAT — MULT.-QUOT. COUNTER — PROGRAM COUNTER

STOP TO TS S1-CP S1-X — CLEAR CY — CYCLE COUNTER — CLEAR $\overline{PC}$ — CLEAR PC AND $\overline{PC}$

NAL TRANSFER BREAKPOINT SELECTOR

CONDITIONAL TRANSFER — TRANSFER — NO TRANSFER — BREAK POINT

– 0 1 2 3 4 5 6 7 8 9

, . ; A B C D E F G H I

SCI CR J K L M N O P Q R

S T U V W X Y Z

OVER FLOW 0 1 2 3 4 5 6 7 8 9 –

SECOND INSTRUCTION DIGIT

INITIAL READ — IR TIMER — CLEAR C — SCI CR

1 2 3 4 5 6 7 8 9 0

BLOCK SUBDIVISION SELECTOR

INPUT SYNCHRONIZER COUNTER — INPUT TANK COUNTER — CLEAR I AND O — OUTPUT SYNCHRONIZER COUNTER — OUTPUT TANK COUNTER — DIRECTION STORAGE — REVERSE — FIRST BLOCK — READ INT. — WRITE INT. — REWIND OVERLOAD

1 2 3 4 5 6 7 8 9 –

INITIAL TAPE SELECTOR

M A X L F C CR SYI EMPTY

OUTPUT SELECTOR

SCI CR — FILL — INPUT READY — SYI1 — 12 TH DIGIT — INPUT ERROR — SYI2 — OUTPUT READY — SKIP — OUTPUT BREAK POINT

ONE OPERATION — ONE STEP — ONE ADDITION — ONE INSTRUCTION

UNIVAC
SUPERVISORY CONTROL PANEL

Remington Rand
ECKERT-MAUCHLY DIVISION

E. J. LEVINSON 9-52

BIBLIOGRAPHY

SEC. 7. BIBLIOGRAPHY

The list of references herein included is not intended to be a complete list but, rather, selected to meet the needs and interests of the reader of this Instruction Manual.

GENERAL REFERENCES

I. 'Preparation of Programs for an Electronic Digital Computer' - Wilkes, Wheeler and Gill (of Cambridge University), Addison Wesley Press, Cambridge, Massachusetts, 1951. (EDSAC)

II. 'High Speed Computing Devices' - Engineering Research Associates Staff, McGraw - Hill, New York, 1950.

III. 'Planning and Coding of Problems for an Electronic Computing Instrument', Goldstine, H.H. and von Neumann, J., Institute for Advanced Study, V. 1, (1947), V. 2, (1948), V. 3, (1948) Princeton, New Jersey.

IV. 'Proceedings of the Symposium on Large Scale Digital Calculating Machinery', Harvard University Computation Laboratory, Cambridge, Massachusetts, Block, R.M. (Mark I) V. 16, 1948.
'Manual of Operation' (Mark I) Chapter I, Chapter IV, 1946 (with complete mathematical literature references, pp 338-404.)
Campbell, R.V.D. (Mark II) V. 16, 1948
'Description of a Relay Calculator' Chapter VII, 1949
'Second Symposium on Large Scale Digital Calculators', 1951
Mauchly J. W., 'Preparation of Problems for EDVAC-Type Machines' V. 16, 1948

V. 'The Binac Manual' A-050-1B dated 7/20/49, Eckert-Mauchly Division of Remington Rand Corp., Philadelphia, Pennsylvania

VI. 'Electronic Digital Computer Papers', University Research Board University of Illinois 1952, Manual 1952.

VII. 'SEAC Operating and Programming Notes', Computation Laboratory U. S. Dept. of Commerce, National Bureau of Standards,
Report I, dated 21 April 1952,
Report II, dated 22 July 1952,
Report III, dated 22 August 1952,
Report IV, dated 12 September, 1952,
Report V, dated 30 September, 1952
'Handbook for SEAC', Joseph H. Levin, 1952

VIII. 'Reference Guide to the UNIVAC System', Prepared for Dept. of Navy ASO, UNIVAC Seminar - January, 1951, Eckert-Mauchly Division, Philadelphia, Pennsylvania

IX. 'Office Robots', Fortune Magazine, pp 82-87 January, 1952
'Mechanical Brains', Fortune Magazine, L.N. Ridenour, V.39 #5 pp 108-118, May, 1949

X. 'Interim Progress Report on the Physical Realization of an Electronic Computing Instrument', Bigelow et al., Institute for Advanced Study,
I Report, January 1947
II Report, July 1947
III Report, January 1948
IV Report, July 1948
V Report, January 1949

XI. 'Calculating Instruments and Machines', University of Illinois, D.R. Hartree, press, Urbana, Illinois 1949
(With 122 references)

XII. 'MIDAC' Papers, Willow Run Research Center, 1952

VII. 'Review of Electronic Digital Computers', Joint AIEE-IRE Computer Conference Publications

XIV. 'Programming for Whirlwind-I' (MIT), H. Saxenian, R-196, 1952 also 'Digital Computers and their Applications' 1952

XV. 'Proceedings of the Association for Computing Machinery', Joint AIEE-IRE, Richard Rimbach Associates, Pittsburg, Pa.

XVI. 'Compiling Routines', Richard K. Ridgway, Paper presented at meetings of the Association for Computing Machinery, September 8, 9, 1952

XVII. 'The Education of a Computer', Dr. Grace M. Hopper - Paper presented at meetings of the Association for Computing Machinery, May 2, 3, 1952

XVIII. 'Solution of Matrix Equations of High Order by an Automatic Computer', Dr. Herbert F. Mitchell, Jr. - Paper presented at the meetings of Association for Computing Machinery, March 28, 29, 1950

XIX. 'High Order Matrix Computation on the UNIVAC', H. Rubenstein and J. Rutledge - Paper presented at the meeting of the Association for Computing Machinery, May 3, 1952

MATHEMATICAL TEXTS

XX. 'Theory and Techniques for the Design of Electronic Digital Computers', Moore School of Electrical Engineering, (EDVAC), Pennsylvania University, 2 V, 1947

XXI. 'Numerical Calculus', W.E. Milne, Princeton University Press, 1949 (Math. methods)

XXII. 'Numerical Mathematical Analysis', J.B. Scarborough, Baltimore, Johns Hopkins Press, 2nd ed., 1950 (Math. methods)

XXIII. 'Interpolation', J.F. Steffensen, Baltimore, Williams and Wilkins, 1927 (Math. methods)

XXIV. 'The Calculus of Observations', E.T. Whittaker and G. Robinson, 4th ed. Glasgow, Blackie and Lon Ltd., 1944

I N D E X

Numbers refer to pages. A page number followed by a (*) means that the reference is concerned with the elementary computer discussed in Chapter 9, although in many cases the reference will be applicable also to UNIVAC.

CHART A

ARITHMETIC AND MEMORY CIRCUITS OF AN ELEMENTARY COMPUTER

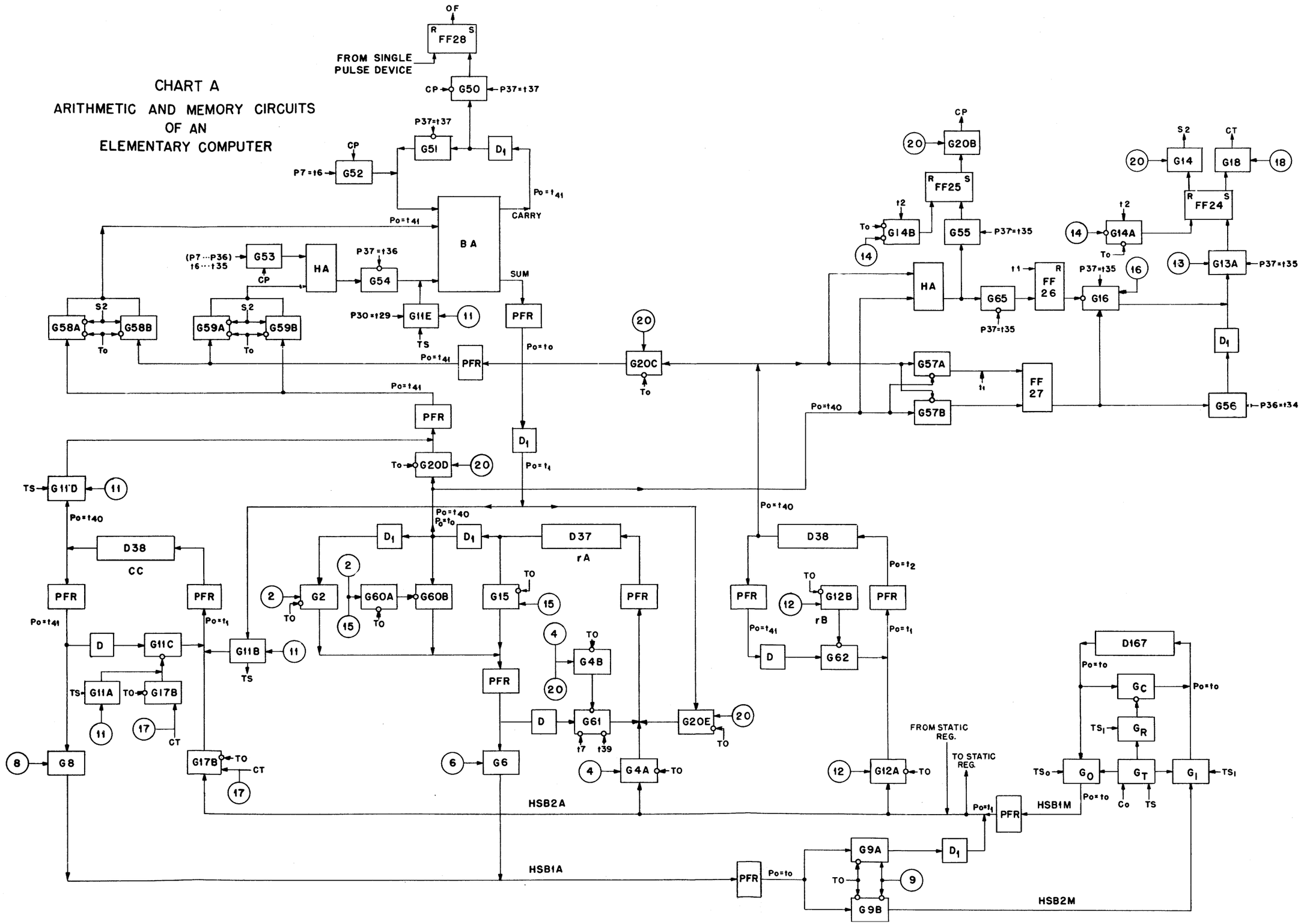


CHART B

CONTROL CIRCUITS
OF AN
ELEMENTARY COMPUTER